NASA Contractor Report 181678

Advanced Information Processing System: Input/Output Network Mangement Software

Gail Nagle
Linda Alger
Alexander Kemp

THE CHARLES STARK DRAPER LABORATORY, INC.
CAMBRIDGE, MA 02139

Contract NAS1-17666
MAY 1988

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure **Title** **Page**

1.0 INTRODUCTION

This purpose of this document is to provide the software requirements and specifications for the Input/Output Network Management Services for the Advanced Information Processing System. This introduction and overview section is provided to briefly outline the overall architecture and software requirements of the AIPS system before discussing the details of the design requirements and specifications of the AIPS I/O Network Management software. Section 1.1 is a brief overview of the AIPS architecture followed by a more detailed description of the network architecture. Section 1.2 provides an introduction to the AIPS system software.

1.1 AIPS Architecture

The Advanced Information Processing System is designed to provide a fault- and damage-tolerant data processing architecture, which can serve as the core avionics system for a broad range of aerospace vehicles, for which NASA has direct or supporting research and development responsibilities. These applications include manned and unmanned space vehicles and platforms, deep space probes, commercial transports, and tactical military aircraft.

AIPS is a multicomputer architecture composed of hardware and software 'building blocks' that can be configured to meet a broad range of application requirements. The hardware building blocks are fault-tolerant, general purpose computers, fault- and damage-tolerant inter-computer and input/output networks, and interfaces between the networks and the general purpose computers (GPCs). The software building blocks are the system software modules: local system services, input/output system services, inter-computer system services and the system manager. This system software provides the traditional services necessary in a real-time computer such as task scheduling and dispatching, communication with sensors and actuators, etc. The software also supplies those services necessary in a distributed system such as inter-function communication across processing sites, management of local and distributed redundancy, management of networks, and migration of functions between processing sites.

The Advanced Information Processing System consists of a number of computers located at processing sites which may be physically dispersed throughout the vehicle. These processing sites are linked together by a reliable and damage tolerant data communication 'bus', called the Inter-Computer Bus (IC) bus. A computer at a given processing site may have access to varying numbers and types of Input/Output (I/O) 'buses' which are separate from the IC bus. The I/O buses may be global, regional or local in nature. Input/Output devices on the global I/O bus are available to all, or at least a majority, of the AIPS computers. Regional buses connect I/O devices in a given region to the processing sites located in their vicinity. Local buses connect a computer to the I/O devices dedicated to that computer. Additionally, I/O devices may be connected directly to the internal bus of a processor and accessed as though the I/O devices reside in the computer memory (memory mapped I/O). Both the I/O buses and the IC bus are time division multiple access contention buses. Figure 1 shows the laboratory engineering model for a distributed AIPS configuration. This distributed AIPS configuration includes all the hardware and software building blocks mentioned earlier and was conceived to demonstrate the feasibility of the AIPS architecture.

The laboratory configuration of the distributed AIPS system shown in Figure 1 consists of four processing sites. Each processing site has a General Purpose Computer. GPCs may be simplex or they may be Fault Tolerant Processors (FTPs) of varying redundancy levels. Of the four FTPs, one is simplex, one is duplex, and two are triplex processors. A FTP may also be quadruply redundant but none was fabricated for the AIPS laboratory demonstration. The redundant FTPs are built such that they can be physically dispersed for damage tolerance. Each of the redundant channels of a FTP could be as far as 5 meters from other channels of the same FTP.

The GPCs are all interconnected by a triplex circuit switched inter-computer (IC) network. Each network layer forms a full two way 'virtual bus' as explained in the next subsection. The three network layers are totally independent and are not cross-strapped to each other. In each network layer there is a circuit switched node for each processing site. Thus every processing site is serviced by three nodes of the IC network. GPCs are designed to receive data on all three layers of the network. The capability of a GPC to transmit on the network, on the other hand, depends on the GPC redundancy level. Triplex FTPs are provided the capability to transmit on all three layers, duplex FTPs on only two of the three layers, and simplex processors on only a single layer. In duplex and triplex FTPs, a given processor can transmit on only one network layer. Thus malicious behavior of a processor can only disrupt one layer.

The IC network and the GPC interfaces into the network are designed in strict accordance with the fault-tolerant systems theory. Thus an arbitrary random hardware fault, including Byzantine faults, anywhere in the system can not disrupt communication between triplex FTPs. In other words, the triplex IC network, in conjunction with the GPC interfaces into the network, provide error-masking capability for inter-GPC communications between triplex computers.

The laboratory demonstration of the Input/Output network is mechanized using a 15 node circuit switched network that interfaces with each of the GPCs on 1 to 6 nodes depending on the GPC redundancy level. The 15 I/O nodes can be configured in the laboratory as global, regional, and local I/O networks to demonstrate various dimensions of the AIPS I/O concept. Further details of the network architecture are described in the following subsection.

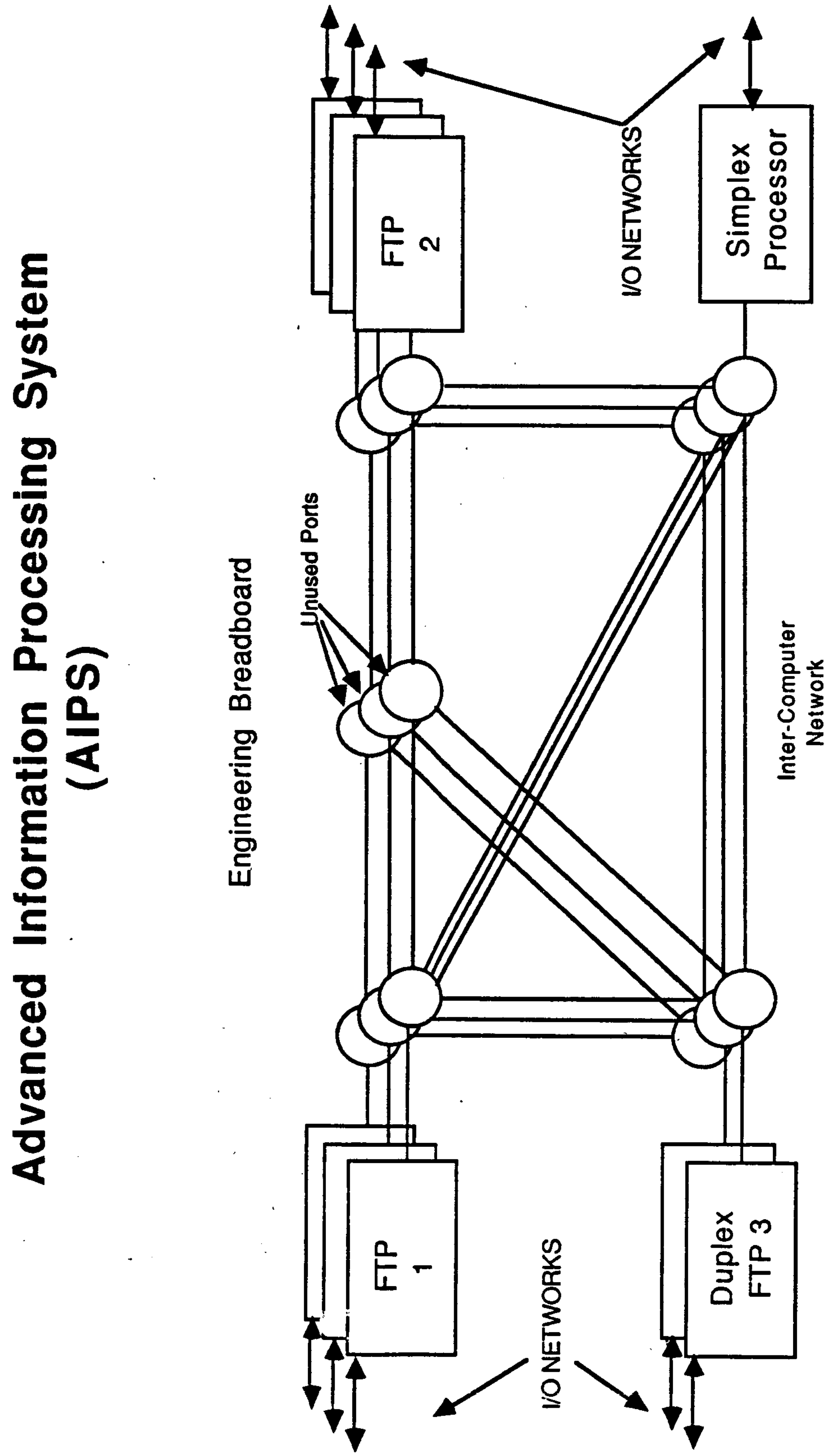


Figure 1. AIPS Distributed Configuration

1.1.1 AIPS Networks

For communication between GPCs and between a GPC and I/O devices, a damage and fault tolerant network is employed. The network consists of a number of full duplex links that are interconnected by circuit switched nodes. In steady state, the circuit switched nodes route information along a fixed communication path, or 'virtual bus', within the network, without the delays which are associated with packet switched networks. Once the virtual bus is set up within the network the protocols and operation of the network are similar to typical multiplex buses. Every transmission by any subscriber on a node is heard by all the subscribers on all the nodes just as if they were all linked together by a linear bus. Although the network performs exactly as a bus, it is far more reliable and damage tolerant than a linear bus. A single fault or limited damage can disable only a small fraction of the virtual bus, typically a node or a link connecting two nodes. Such an event does not disable the network, as would be the case for a linear bus. The network is able to tolerate such faults due to the richness of interconnections between nodes. By reconfiguring the network around the faulty element, a new virtual bus is constructed. Except for such reconfigurations, the structure of the virtual bus remains static.

The nodes are sufficiently intelligent to recognize reconfiguration commands from the network manager, which is resident in one of the GPCs. The network manager performs the necessary diagnostics to identify the failed element and can change the bus topology by sending appropriate reconfiguration commands to the affected nodes.

Damage caused by weapons or electrical shorts, overheating, or localized fire would affect only subscribers in the damaged portion of the vehicle. The rest of the network, and the subscribers on it, can continue to operate normally. If the sensors and effectors are themselves physically dispersed for damage tolerance, and the damage event does not affect the inherent capability of the vehicle to continue to fly, then the digital system would continue to function in a normal manner or in some degraded mode as determined by sensor/effector availability.

Fault isolation is much easier in the network than in multiplex buses. For example, a remote terminal transmitting out of turn, a rather common failure mode which will totally disable a linear bus, can be easily isolated in the network through a systematic search where one terminal is disabled at a time. Furthermore, for networks of moderate size, up to 50 nodes, most faults can be detected, isolated and the network reconfigured in milliseconds.

The network can be expanded very easily by linking the additional nodes to the spare ports in existing nodes. In fact, nodes and subscribers to the new nodes (I/O devices or GPCs) can even be added without shutting down the existing network. In bus systems, power to buses must be turned off before new subscribers or remote terminals can be added.

Finally, there are no topological constraints, as are encountered with linear or ring buses. In fact, these are simply subsets of the fault-tolerant network architecture.

1.2 AIPS System Software

The AIPS system software along with the hardware has been designed to provide a virtual machine architecture that hides hardware redundancy, hardware faults, multiplicity of resources, and distributed system characteristics from the applications programmer. The following section, 1.2.1, is a discussion of the approach that is used for the AIPS system software design. Section 1.2.2 presents a brief high level description of the AIPS system services that are provided for the AIPS system user.

1.2.1 AIPS Software Design Approach

The approach used to design the AIPS system software is part of the overall AIPS system design methodology. An abbreviated form of this system design methodology is shown in Figure 2. This methodology began with the application requirements and eventually led to a set of architectural specifications. The architecture was then partitioned into hardware and software functional requirements. This report documents the software design approach starting from functional requirements, to software specifications, to Ada implementation as applied to Input/Output Network Management software. The I/O Network Management software is a part of the I/O System services.

Hardware and software for the AIPS architecture is being designed and implemented in two phases. The first phase is the centralized AIPS configuration. The centralized AIPS architecture, as shown in Figure 3, is configured as one triplex Fault Tolerant Processor (FTP), an Input/Output network and the interfaces between the FTP and the network, referred to as input/output sequencers (IOSes). The laboratory demonstration of the input/output network consists of 15 circuit switched nodes which can be configured as multiple local I/O networks connected to the triplex GPC. For example, the I/O network may be configured as one 15 node network as shown in Figure 3, or as three 5 node networks. The software building blocks that have been designed and implemented for the AIPS centralized architecture include local system services and I/O system services. The following subsection 1.2.2 will give an overview of all the AIPS software building blocks. The rest of this document , Sections 2 thru 6, focuses on the software designed for the redundancy management of the I/O networks.

1.2.2 AIPS System Software Overview

AIPS system software provides the following AIPS System Services (Figure 4): local system services, communication services, system management, and I/O system services The system software is being developed in Ada. System services are modular and naturally partitioned along hardware building blocks. The distributed AIPS configuration includes

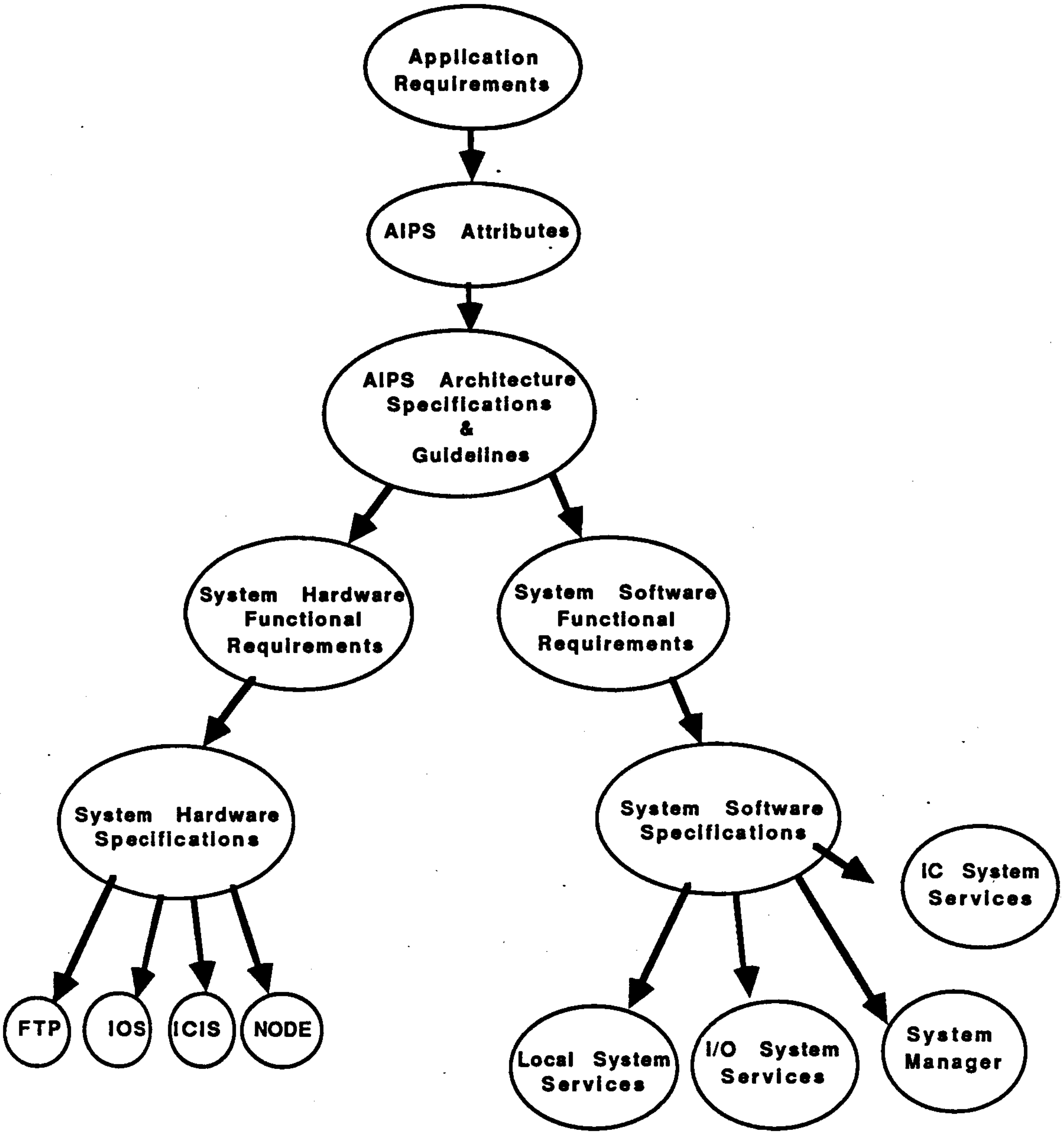

Application Requirements
AIPS Attributes
AIPS Architecture Specifications & Guidelines
System Hardware Functional Requirements
System Software Functional Requirements
System Hardware Specifications
System Software Specifications
IC System Services
FTP
IOS
ICIS
NODE
Local System Services
I/O System Services
System Manager

Figure 2. AIPS System Design Approach

all the services. Specific versions of the system software for different applications can be created by deleting services from this superset. Shared resource allocation and redundancy management are implemented only once but not necessarily in the same General Purpose Computer (GPC). The other system services are replicated in each GPC. The following is a brief description of each of the services.

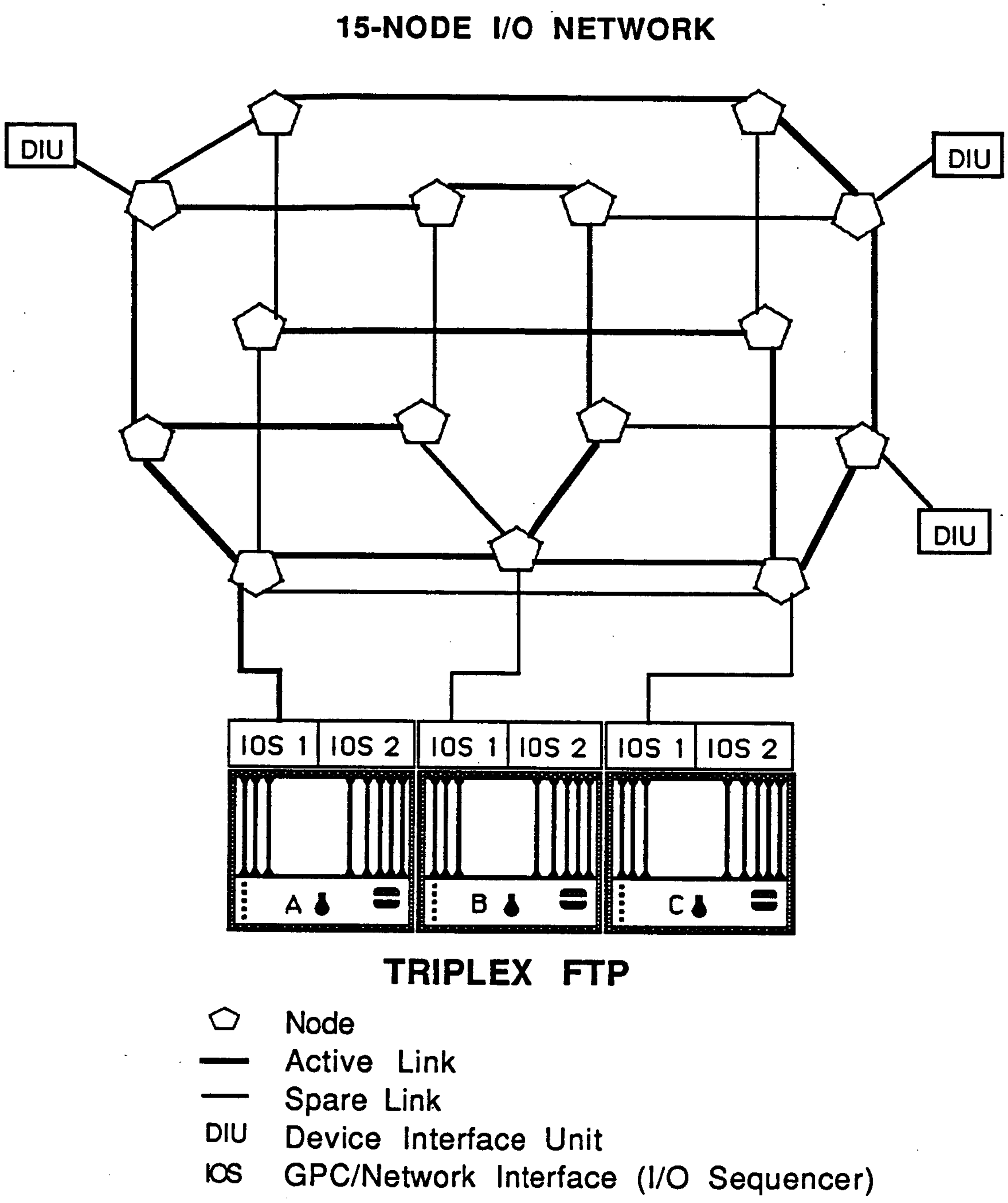


Figure 3. Centralized AIPS Configuration

1.2.2.1 Local System Services

The local system services provided in each GPC are: GPC initialization, real-time operating system, local resource allocation, local GPC Fault Detection, Isolation, and Reconfiguration (FDIR), GPC status reporting, and local time management (Figure 5).

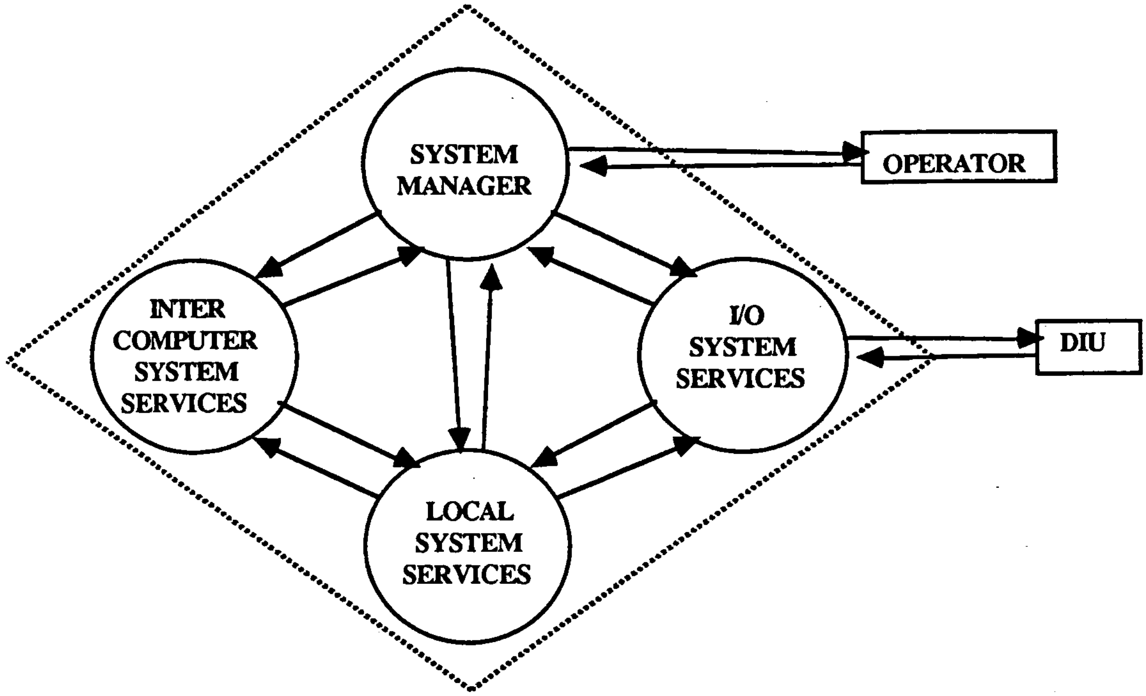


Figure 4. Top Level View Of System Services

The function of GPC initialization is to bring the GPC to a known and operational state from an unknown condition (cold start). GPC initialization synchronizes the CPs, synchronizes the IOPs and resets or initializes the GPC hardware and interfaces (interval timers, real time clock, interface sequencers, DUART, etc.) It makes the hardware state of the redundant channels congruent by alignment of memory and control registers. It then activates the system baseline software that is common to every GPC.

The AIPS real-time operating system supports task execution management including scheduling according to priority, time and event occurrence, and is responsible for dispatching, and task suspension and termination. It also supports memory management, software exception handling and intertask communication between the companion processors (IOP and CP). The AIPS operating system is resident on every CP and IOP in the system. It uses the vendor supplied Ada Run Time System (RTS), and, in addition, provides those extensions necessary for the AIPS real-time distributed operating system.

The GPC resource allocator coordinates and determines responsibility for any global or migratable functions from the system resource manager. It also monitors commands from the system resource manager to start or stop any function.

The GPC status reporter collects the status information from the local functions, the local

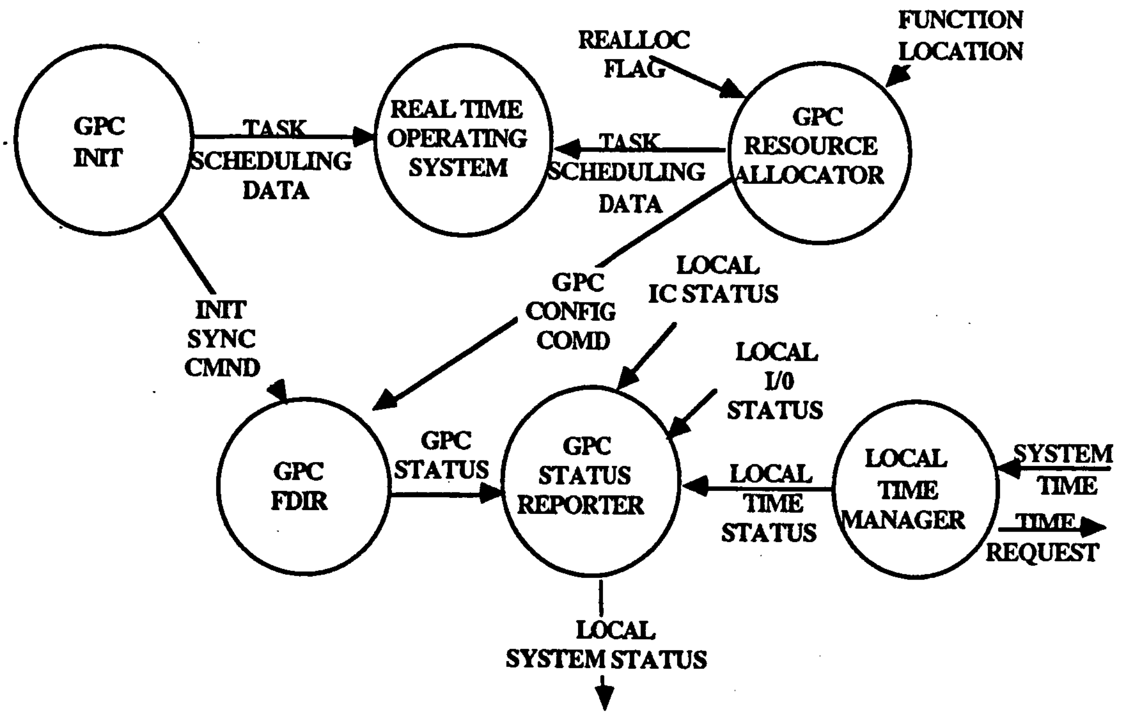


Figure 5. Local System Services

GPC FDIR, the local time manager, the IC system services and the I/O system services. It updates its local data base and disseminates this status information to the system manager.

The GPC FDIR has the responsibility for detecting and isolating hardware faults in the CPs, IOPs, and shared hardware. It is responsible for synchronization of the redundant channels of the bi-processor FTP, and for disabling outputs of failed channel(s) through interlock hardware. Since each channel of an FTP has two processors (bi-processor), the synchronization software is responsible for the tight synchronism of both redundant groups of processors. After synchronization, all CPs are executing the same machine language instruction within a bounded skew, and all IOPs are executing the same machine language instruction within a bounded skew. GPC FDIR logs all faults and reports status to the GPC status reporter. It is responsible for the CPU hardware exception handling and downmoding/upmoding hardware in response to configuration commands from the system manager. It is also responsible for transient hardware fault detection and for running self tests at the lowest priority in order to detect latent faults. This redundancy management function is transparent to the application programmer.

The local time manager works in cooperation with the system time manager to keep the local real time initialized and synchronized to the global real time. It updates local offset in response to time broadcasts from the system time manager. It is responsible for reading the real time clock and providing time services to all users.

1.2.2.2 Inter-Computer Services

The inter-computer services provide two functions: inter-computer (IC) user communication services, that is, communication between functions not located in the same GPC; and the IC network management (Figure 6).

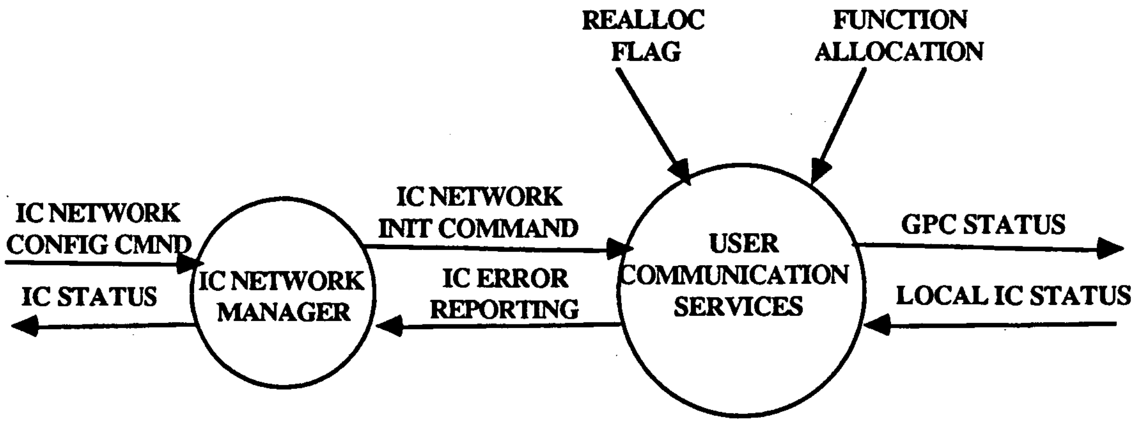


Figure 6. Inter-Computer Services

The IC user communication service provides local and distributed inter-function communication as a transparent service to the application user. It provides synchronous and asynchronous communication, performs error detection and source congruency on inputs, records and reports IC communication errors to IC network managers. Inter-computer communication can be done in either point to point or broadcast mode and is implemented in each GPC.

The IC network manager is responsible for the fault detection, isolation and reconfiguration of the network. The AIPS distributed configuration consists of three identical, independent IC network layers which operate in parallel to dynamically mask faults in a single layer and provide reliable communication. There is one network manager for each network layer. However, the three network managers do not need to reside in the same GPC. They are responsible for detecting and isolating hardware faults in IC nodes, links and the IC interface sequencer and for reconfiguring their respective network layer around any failed elements. The network manager function is transparent to all application users of the network.

1.2.2.3 System Manager

The system manager is a collection of system level services including the applications monitor, the system resource manager, the system fault detection, isolation and reconfiguration (FDIR), and the system time manager (Figure 7).

The applications monitor interfaces with the applications programs and the AIPS system operator. It accepts commands to migrate functions from one GPC to another, to display system status, to change the state of the system by requesting a hardware element state change, and to convey requests for desired hardware and software configurations to the system resource manager.

The system resource manager allocates migratable functions to GPCs. This involves the monitoring of the various triggers for function migration such as failure or repair of hardware components, mission phase or workload change, operator or crew requests and timed events. It reallocates functions in response to any of these events. It also designates managers for shared resources and sets up the context manager data base in each GPC.

The system fault detection, isolation and reconfiguration (FDIR) is responsible for the collection of status from the inter-computer (IC) network managers, the I/O network managers, and the local GPC redundancy managers. It resolves conflicting local fault isolation decisions, isolates unresolved faults, correlates transient faults, and handles processing site failures.

The system time manager along with each GPC local time manager has the job of maintaining a common timebase throughout the system. The system time manager indicates to the local time manager when to set its value of time. It also sends a periodic signal to enable the local time manager to adjust its time to maintain synchronism with an external time source such as the GPS Satellites or an internal source such as the real time clock in the GPC which hosts the system time manager software.

1.2.2.4 I/O System Service

The I/O system service provides efficient and reliable communication between the user and external devices (sensors and actuators). The I/O system service is also responsible for the fault detection, isolation and reconfiguration of the I/O network hardware and GPC/network interface hardware (input/output sequencers).

I/O system service is made up of three functional modules: I/O user interface, I/O communication management and the I/O network manager (Figure 8).

The I/O user interface provides a user with read/write access to I/O devices or Device Interface Units (DIUs), such that the devices appear to be memory mapped. It also gives the user the ability to group I/O transactions into chains and I/O requests, and to schedule I/O requests either as periodic tasks or on demand tasks.

The I/O communication manager provides the functions necessary to control the flow of data between a GPC and the various I/O networks used by the GPC. It also performs source congruency and error detection on inputs, voting on all outputs, and reports communication errors to the I/O Network Manager. It is also responsible for the management of the I/O request queues.

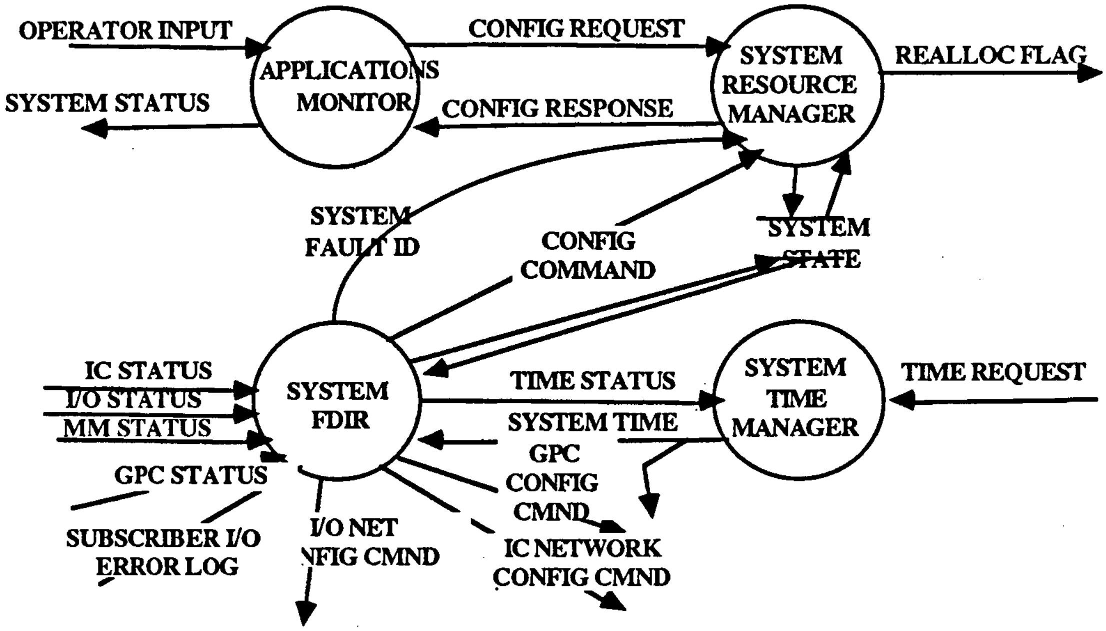


Figure 7. System Manager

The I/O Network Manager is responsible for detecting and isolating hardware faults in I/O nodes, links, and interface sequencers. The I/O network manager is also responsible for reconfiguring the I/O network around any failed elements. The network manager function is transparent to all application users of the network. Section 2 describes the functional requirements and algorithms used for the network management software. Section 3 is the software specification of the network management software, and Section 4 is a detailed description of the Ada implementation of this software. Finally, Section 5 concludes with a summary of results and suggestions for future work in this area.

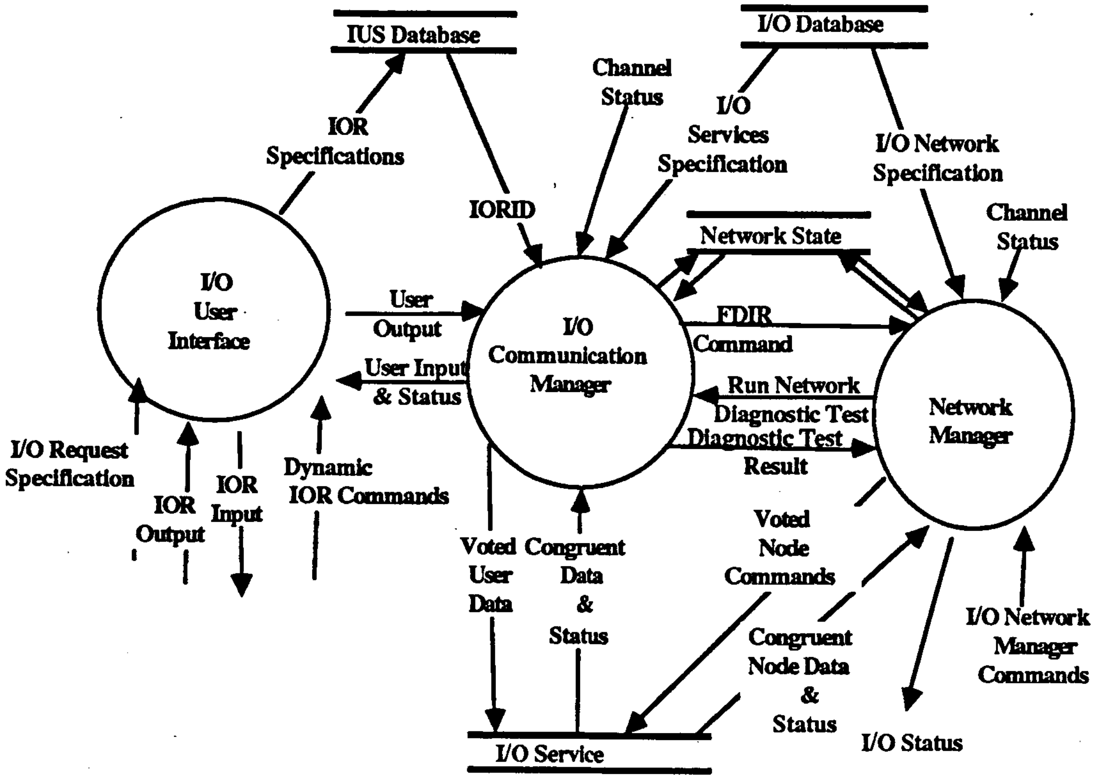


Figure 8. I/O System Services

2.0 I/O NETWORK MANAGER FUNCTIONAL REQUIREMENTS

2.1 Introduction

AIPS Input/Output Networks are briefly discussed in Section 1.1.1. Figure 9 shows an AIPS configuration highlighting the features of an I/O Network. The figure shows an AIPS system in which two GPCs are physically connected to an I/O Network. Each GPC is connected to the network by means of two root links. The input/output operations on the network are conducted by the I/O Sequencer (IOS) which is controlled by the GPC through the Dual Ported Memory (DPM). The network shown consists of six nodes and four Device Interface Units (DIUs). Sensors, actuators, displays and other I/O devices are attached to the DIUs. The network exists to allow application programs executing on the GPC to communicate in a highly reliable manner with these I/O devices. The high reliability of the network is due to the fact that when a hardware component fails or is damaged by some external event, the failed component can be isolated from the rest of the network and communication can proceed along a new path in the network.

In the steady state, the communication path operates as if it were a conventional, time division multiplex bus. It differs from a linear bus in that the data is routed by circuit switched nodes along one of several possible paths. Each node in a properly configured, fault free network receives transmissions on exactly one of its enabled ports and then retransmits this data from all its other enabled ports. Since the nodes are circuit switched, the incoming data is not buffered. Hence, the network does not suffer from the transmission delays associated with packet switched networks. The nodes provide a richness of spare interconnections which can be brought into service after a hardware fault or damage event occurs. The network architecture provides coverage for many well known failure modes which would cause a standard linear bus to either fail completely or provide service to a reduced subset of its subscribers. These failure modes include component failures which result in babblers, i.e. subscribers which use the network in violation of established turn-taking protocols and physical damage events which result in severed cables or other component loss.

Once a properly functioning virtual bus has been established, the nodes used to form the bus remain in the active network until a component fails or is damaged. The configuration of these nodes varies slowly over time to allow spare links to be brought into active service. In response to failures, this process reconfigures the communication path to exclude the failed component.

The ability to reroute data along different paths comes from the design of the node. An AIPS node has five ports which can each be enabled or disabled. When the ports on either end of a link are enabled, data is routed along that link of the network. In Figure 9 the active links, i.e. those connecting two enabled ports, are shown as solid lines. The links shown as dashed lines are spares. A message transmitted by the IOS in Channel A of

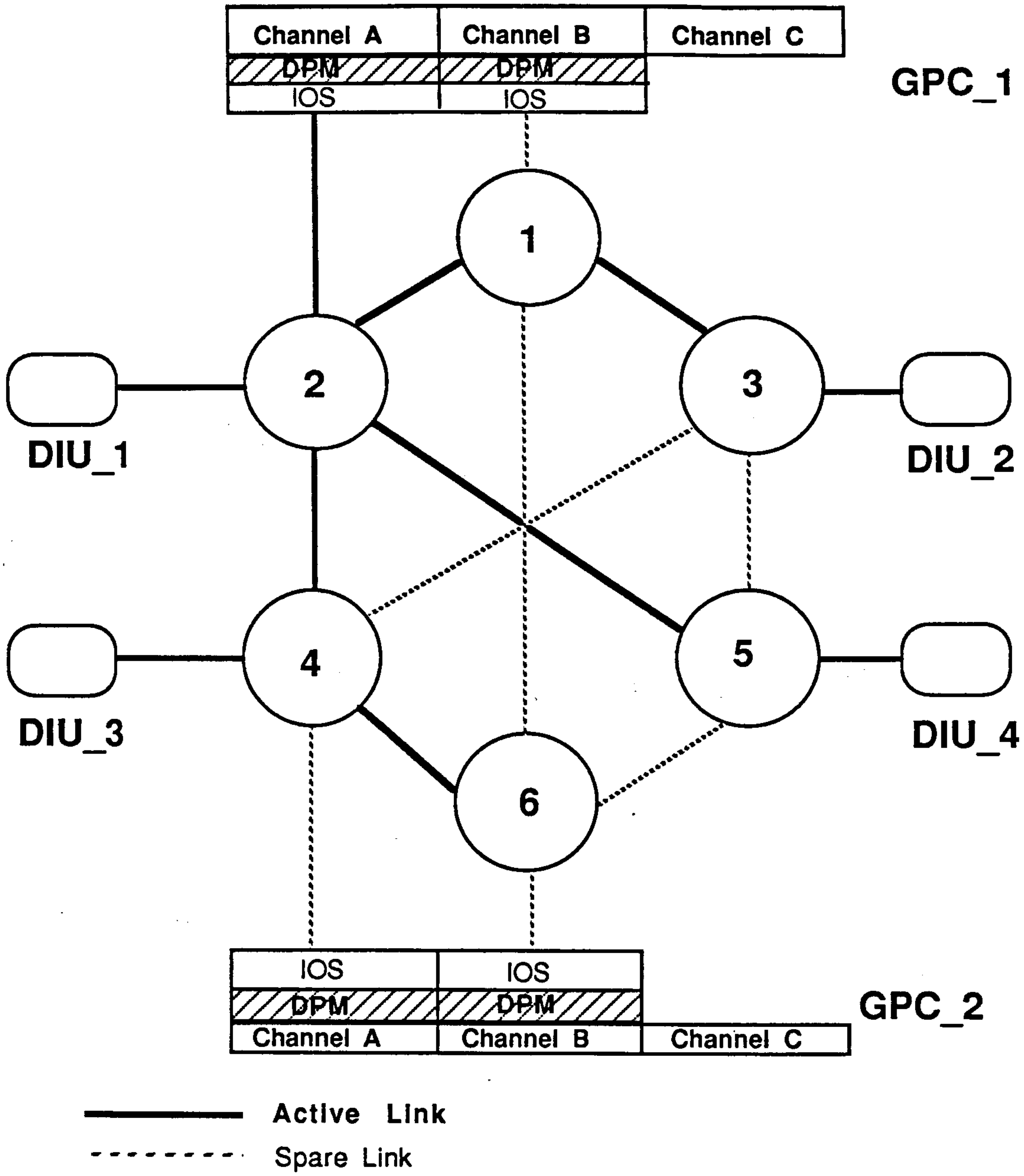


Figure 9. I/O Network With Root Links To Two GPCs

GPC_1 would first reach Node 2. From there it would be simultaneously retransmitted to Nodes 1, 5 and 4. From Node 1 it would be retransmitted to Node 3 but not to Node 6 since the link between Nodes 1 and 6 is a spare. From Node 3, the message would be retransmitted to DIU_2 only. If the link connecting Nodes 1 and 2 were to be severed, thereby interrupting service to DIU_2, the spare link between Nodes 3 and 4 could be enabled to restore full service to all the DIUs on the network.

Another feature of the topology shown in the figure is the fact that GPC_2 is not actively connected to the network. This is because the network shown is a local network, one whose use is dedicated to a single GPC. However, if faults were to cause a degraded mode of operation for GPC_1, the functions requiring access to the network could be migrated to GPC_2. The physical connections to GPC_2 are provided to support function migration. However, this topology is also capable of supporting a regional network, i.e. one which is shared among several GPCs simultaneously. If this were a regional network, GPC_2 would have an active root link to the network and both GPCs would then share the resources of this network by contending for its use.

2.2 I/O Network Manager Interface to AIPS System Services

The I/O Network Manager is the software process responsible for establishing and maintaining a communication path between processors (GPCs) and DIUs attached to the I/O network under its control. Figure 10 presents a high level view of the I/O Network Manager in relation to other software processes with which it interacts. The shaded regions indicate the subprocesses of I/O System Services with which the I/O Network Manager must interact. Non shaded regions are processes in other System Services. In particular, the Resource Allocator is a subprocess within the System Manager and GPC FDIR and the Operator Interface are part of Local System Services.

The Network Manager has two phases of operation: initialization and maintenance. When the Network Manager is called by the Resource Allocator to manage a network, it enters its initialization phase of operation. The Manager's activity during this phase of operation is dictated by the reason for its activation. If the Manager is activated to manage a previously inactive network, or when a graceful function migration is not possible, the Manager establishes a virtual bus within the network and performs a full set of diagnostic tests on each IOS and nodal port in the network. At the end of this initialization process, a fully tested communication path exists between all properly functioning nodes, DIUs, and GPCs in the network. This path is then capable of supporting serial communication among all functioning network subscribers. If the migration of a Network Manager from one GPC to another can be effected gracefully, data from the deactivated Manager is transferred to the newly activated Manager. Thus, if the Manager is activated as part of a graceful function migration, the initialization phase can be reduced to a software component only, followed by a diagnostic test of the existing network configuration. Network reconfiguration will only be necessary if this test uncovers faults in the network.

Having completed its initialization, this process notifies I/O Communication Services that the network is in service and updates the status information on this network which is available to other processes in the system. A potential user of this information is the System Manager. The Network Manager then enters the maintenance phase of its operation.

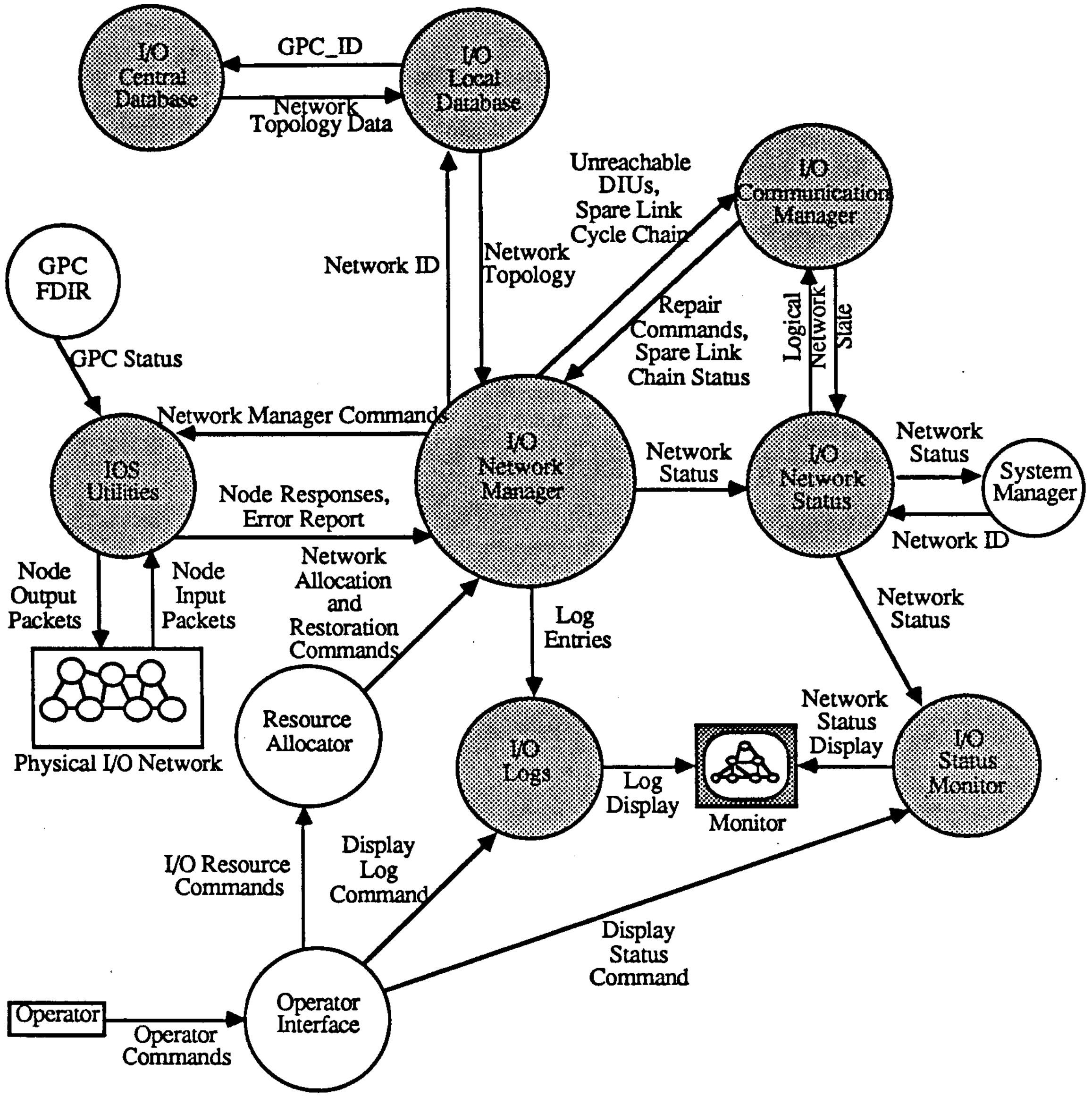


Figure 10 Data Flow Diagram of I/O Network Management Software

During the maintenance phase of its operation, the Network Manager provides services on demand to the Resource Allocator and to the I/O Communication Manager. The Resource Allocator calls this process when it wishes to halt the management of this network from this GPC. This may be to effect a function migration or to support routine system maintenance. The I/O Communication Manager calls this process for one of three reasons: to repair a suspected network fault, to bring a repaired node, link or IOS back online, or to routinely retire an active link and replace it with a spare link. The last operation is called spare link cycling.

Although networks which are grouped together to form an I/O Service are operated in parallel by the I/O Communications Manager when executing user chains or spare link tests, this is not the case for a Network Manager. The I/O Communications Manager controls simultaneous I/O activities on a set of networks in an I/O Service, but the I/O Network Manager has access to only one network. The Network Manager is responsible for network maintenance, that is, for reconfiguring the network in response to a fault. It must be possible to reconfigure a network so that a failure in one member of an I/O Service can be repaired without inhibiting communications in the other members which do not have faults. Thus the operation of each Network Manager is completely independent of the operation of any other Network Manager in the system. This feature is supported by the protocol between the I/O Network Manager and the I/O Communication Manager. When a network is being maintained, it is under the exclusive control of the Network Manager. However, other networks in the I/O Service remain under the exclusive control of the Communications Manager.

The protocol between the Network Manager and the I/O Communication Manager to effect a network repair operates as follows. Whenever the I/O Communication Manager detects a communication error while using the network to conduct normal I/O operations for application processes, it takes that network out of service and calls the Manager of that network to repair the network. The I/O Communication Manager will not use this network until the Network Manager has indicated that the network is repaired. When the Network Manager is scheduled in response to a request from the I/O Communication Manager for network maintenance, it becomes the sole user of the network until the repair is complete. The Network Manager first executes a chain to collect some real time data from the network. The chain requests each node in the network to report its status. The node should respond with its current port configuration and the type of activity each port has seen since the last time its status was read. This monitoring does not alter the node configuration. The node status reports are processed to determine what type of failures, if any, are present in the network.

The Network Manager can detect and repair the passive failure of a node or port, the passive failure of an IOS, the failure of the channel connected to the active root link, a network component which is babbling, a node which answers to addresses other than its own, or a node which transmits on a disabled port. Once the failure mode has been determined, an appropriate maintenance procedure is executed. The network is reconfigured to remove the faulty component and restore communication to all non-failed components in the network. Maintenance procedures are designed to reconfigure the network in the fastest possible manner so as to restore use of the network to application users as quickly as possible. After the reconfiguration of the network is complete, some DIUs may be unreachable. A list of these unreachable DIUs is made available to the I/O Communication Manager when the network is put back in service. This enables it to deselect transactions to unreachable DIUs and to clear error counts against I/O devices which were temporarily out of service due to network problems.

Spare link cycling employs a different protocol between the Network Manager and the I/O Communication Manager. The Network Manager prepares a set of commands which can be sent to the nodes whenever the network has available bandwidth for this operation. This set of commands is sent to the nodes at the discretion of the I/O Communication Manager who is also in charge of using the network to communicate with devices for application programs. Thus, the network is not taken out of service to conduct this test and the results of the test are processed by the Network Manager when there is available processing time on the system for this purpose. A semaphore mechanism enforces mutual exclusion between the two Managers for data access privileges to the node commands. All spare links, including root links, are routinely cycled to determine whether or not they are operating properly and can therefore be reliably called into service to reconfigure the network after a failure of some active link. Cycling spare links provides greater fault coverage than merely testing a link and then restoring the active link to service since all parts are exercised for longer periods of time. Spare links are cycled at a rate commensurate with the desired fault detection latency and the testing overhead.

The protocol used by the two Managers when restoring a failed link, node or IOS to service depends on whether or not the restoration can be effected without transmitting messages to nodes on the network. In the case of a link, the restoration does not require the transmission of node commands on the network. Hence, the network is not taken out of service during a restoration of this type. However, a node or an IOS can only be restored by reconfiguring the network, and this can only be accomplished by using the network to transmit commands to nodes. Thus, these types of restorations do require that the network be taken out of service during the restoration and returned to service once it is complete.

It should be noted that the procedure for restoring failed components in the present Network Manager design is not automatic. Once a component has been declared failed, it remains out of service until the Network Manager is requested by the operator (via the I/O Communication Manager) to restore the failed component. The Network Manager does not test failed components periodically to determine if they have been repaired.

2.3 I/O Network Manager Interface to Subprocesses in I/O System Services

The I/O Network Manager is supported in its operation by various subprocesses within I/O System Services. These are shown in Figure 10 as shaded regions. The function of each of these subprocess is discussed here.

2.3.1 I/O Sequencer Utilities

The operation of the I/O Sequencer (IOS) is described in detail in the IOS Specification (Appendix D). The IOS is a specialized hardware unit designed to have direct access to the serial I/O Networks of an AIPS system. Its primary purpose is to offload the IOP from the work needed to transmit and receive data. While it operates asynchronously from the IOP, it is nevertheless under software control. This control is exerted through a set of registers which allow the IOP to know the state of the IOS and to issue commands to the IOS. The IOS operates by executing a program which the IOP has previously stored in the Dual Ported Memory (DPM) shared by the IOP and the IOS. It transmits data which also has been stored in the DPM and in turn stores incoming data from the network in the DPM. The IOS is also able to contend for network use for its IOP. Under no fault conditions, transmissions to network nodes are always followed by responses from the nodes. A transmit/response pair is called a transaction. A set of transactions which are grouped together sequentially for rapid and uninterrupted transmission on the network is called a chain. The IOS is said to execute a chain of transactions when it contends for network use and then executes a program which sends and receives data on the network without interruption from any other GPC subscriber to the network. The Network Manager configures the network nodes by using the IOS to execute chains of transactions which communicate to the nodes. It also uses the error detection capabilities of the IOS to help it diagnose faults in the network.

Since primary function of the IOS is offloading the IOP from the low level aspects of serial communications, it is undesirable to reload the dual ported memory with IOS programs and static data, i.e. data which does not change for each chain execution since this takes IOP processing time. Hence, the dual ported memory is organized to hold all necessary programs and data used by the IOS.

2.3.2 I/O Databases

The I/O Network Databases serve as a repository of static information about I/O networks. They contain a software description of the physical makeup of the I/O networks in the system. They also contain the information necessary to map logical information related to networks into its physical counterpart. For example, the logical identifier of a DIU would be mapped to its physical address on a network. In the present implementation, the baseline topology of an I/O network does not change in real time, i. e. the number of nodes, DIUs, and GPCs in a network and the physical interconnections between them is fixed at run time. Hence the information in the databases also does not change in real time either. The databases also contain information about the organization of the I/O networks into I/O Services.

The I/O Central Database holds information about every I/O network and every I/O Service in the system. It is intended to support the use of mass storage which is accessible from every GPC in the system. While there is only one logical I/O Central Database in any AIPS system, an I/O Local Database resides on every GPC but only contains information about the I/O networks to which that GPC is physically connected. When no mass storage device is included in an AIPS architecture, the I/O Central Database will reside on every GPC. It should be noted that there is no duplication of data in this case; the I/O Local Database references the I/O Central Database directly.

The I/O Local Database references the I/O Central Database during program initialization to obtain information about the networks to which its GPC is connected. Since one characteristic of mass storage devices is a long access time, these accesses may be relatively slow. Because speed is not important during system initialization, this slow access time is not a problem. Using this information, the I/O Local Database deduces other information about its networks and stores all this data locally. Deducing information about networks whenever possible from more fundamental data has some advantages. It reduces the amount of information that must be hand generated. This type of data entry is laborious, and therefore costly. It is also error prone; thus the derived data is more reliable. When another process in the GPC needs information about an I/O network, it will obtain this information from the I/O Local Database. Unlike data retrieval from mass storage, these memory accesses will be very fast.

2.3.3 I/O Network Status

I/O Network Status serves as a repository of information about the state of every network in the system. The state of a network is comprised, in part, of the most current information about the condition of its hardware as well as other facts which can be deduced from the state of its hardware. Furthermore, since the network is a shared physical resource under software control, the state is also comprised of information about which process has access to the network at any given time, which IOS is active and which DIUs are reachable. Two processes share responsibility for determining network status: the I/O Network Manager and the I/O Communication Manager. Other processes which may be consumers of this information include the Resource Allocator, the GPC Status Reporter, the System Manager, and the I/O Network Status Monitor.

The hardware components in the network which are viewed as part of the AIPS system are the nodes, the ports of the node, and the IOSs. (A link is defined as two ports on adjacent nodes and the cable between them). While DIUs are physically part of the network, they are not considered part of the underlying system but rather part of the application process. In general, I/O System Services would not have enough information to determine whether or not a given sensor, actuator, or other DIU component was functioning properly. What I/O System Services can determine is whether or not it is possible to carry on error free communications with a given DIU, where errors are defined as any violations of the strict protocol which governs such communications. The state of the nodes and the IOSs is determined solely by the Network Manager. The hardware status is the Network Manager's view of the network hardware made available to any other process in the system. Of course, the actual physical state of the hardware may change many times during network growth and reconfiguration. However, these transitionary periods are of short duration. Therefore, the values stored by this process are stable values representing the view of the Network Manager after any necessary changes in configuration have been made. The state of DIUs, the rootlink currently in use, and who controls access to network resources is determined jointly by the I/O Communication Manager and the I/O Network Manager.

2.3.4 I/O Network Logs

While I/O Network Status records information for use by other software processes in the system, there is a clear need for information about the status and history of network hardware to made available to a human operator. Such information can be useful for online system maintenance. It may also be an important input into mission critical decisions which are under operator control. This process is responsible for keeping a log relating to the history of network hardware for each network in the system. The Network Manager and the I/O Communication Manager both make log entries. An operator is able to display those entries on a terminal.

2.3.5 I/O Network Status Monitor

Since it is helpful for the I/O network status to be easily visible to an operator, a network status display is provided for each I/O network in the AIPS system. This status of the AIPS I/O networks may be displayed on a VT100 terminal or a VT240 color terminal, depending on availability.

The display is derived from the Network Manager's view of the status of the network hardware. The display process periodically queries the I/O Network Status process about changes in the status of the network. If changes have occurred since the last time the display process obtained status information, it updates the display accordingly. The display is not completely redrawn each time network status changes which produces a significant gain in the response time of the display.

2.4 Algorithmic Considerations

2.4.1 Network Growth

Network growth is the process whereby the links between the nodes in the network are enabled to form a virtual bus which supports communication among network subscribers (GPCs and DIUs). Data flow in the network is controlled by the configuration of the ports in each node. For a link to carry data between two nodes, the ports at either end of the link must both be enabled. Nodes retransmit messages received by an enabled port from its other enabled ports, but not from the port which received the message. (The purpose of the retransmission is to maintain the integrity of the waveform and only imposes a delay of one half the transmission clock period.) When a node receives a message addressed to itself on any port, disabled or enabled, it carries out the command encoded in the message and then transmits its status from all its enabled ports, including the port which received the message if that port is enabled. A node obeys reconfiguration commands sent by the Network Manager by enabling or disabling its ports in accordance with the value of the port enable field in the command. Once the new configuration is in effect, the node returns a status message. There are no restrictions in the overall network topology. However, for proper operation, there can be no loops in the active network. The nodal ports which are enabled may not establish a ring. A data bit travels through each enabled link exactly once. Once it is grown, a network operates like a time division multiplex bus.

Nodes are added one by one to the virtual bus. To determine which node to add next, the Network Manager refers to the Network Topology, a database which describes all the physical interconnections which exist in the network on a node by node basis. The algorithm used to add these nodes causes the bus to expand in a treelike manner. Because of its resemblance to a tree, the nodes which are included as part of the virtual bus are said to be part of the active tree. The growth algorithm generates a maximally branching, minimum length path to every node in the network. This configuration is later changed in order to cycle spare links and to repair faults. In addition to joining network nodes into a virtual bus, the growth process is also concerned with enabling communication paths to network subscribers: DIUs and remote GPCs. This is accomplished by enabling nodal ports adjacent to these devices and determining whether or not these components obey the protocols established for all functioning network components. The detection of protocol violations results in the connection to the subscriber being disabled. In fact, the detection of a protocol violation when any new link is called into service results in the disabling of that link. Furthermore, the growth algorithm employs a set of diagnostic tests which exercise every link in the network, including spare links. The tests can also detect the presence of some malicious failure modes such as nodes which transmit on disabled ports and nodes which respond to commands addressed to other nodes.

The network growth algorithm assumes that, although hardware faults may be present in the network before the growth process commences, no additional faults will occur while growth is taking place. However, if errors are detected during growth which indicate an additional failure, then the growth process begins again from the top. Failure modes which produce this condition are the failure of the active root link, or the presence of a babbler. If a fault occurs repeatedly after a network is partially grown, an intermittent failure can be inferred. Strategies to deal with short lived, intermittent failures need to be developed. However, this is beyond the scope of this functional design.

Network growth begins by establishing an active root link to one of the root nodes and ensuring that this root node has a port which can be used as the springboard to the rest of the nodes in the network. If an active root link is found, the remaining nodes are added to the active tree. Any nodes which are not connected to the active tree after this stage is complete are unreachable. At this point in the growth process only one root link to the network has been enabled. After the nodal network is established through the active root link, the spare root links to the network must be enabled and tested. In order to establish spare root links, the inboard port of each active root node is enabled. Next the network subscribers, that is the DIUs and remote GPCs of a regional network, are connected to the network. Finally, status is collected from all nodes in the network to verify that no failures have occurred in the network during the growth process. If no discrepancies are found, the node status chain is updated by removing transactions to nodes which have been identified as failed. Figure 11 summarizes the major steps in the network growth algorithm. The following discussion examines the logic employed in each major step in more detail.

```
Repeat until growth is successful or two attempts fail to produce a stable network:
    Establish a working connection to a root node
    If an active root link is established then
        Add remaining nodes to the network
        Mark idle nodes failed
        Add spare root links
        Add DIUs
        Add Remote GPCs
        Collect Node Status from all nodes in network as defined by topology
        Validate Network Status
        If no discrepancies in Network Status then
            network is grown successfully
            deselect transactions of failed nodes from the status collection chain
```

Figure 11. The Network Growth Algorithm

For growth of a network to be considered successful, an active root link must connect the GPC to the network. This implies the existence of a properly functioning IOS and, except in the trivial case of a one node network, a root node which is able to communicate not only with the IOS but also with at least one adjacent node. Establishing the connection is a two step procedure. In the first step, the hardware is put in a state which supports communication between the GPC and the root node. In the second step, the correct operation of this hardware is verified. In an optional third step, a set of diagnostic tests is performed.

Since a GPC generally has more than one root link to a network, the approach taken is to order the root links in some way and then to try them in turn until a properly functioning connection is found. The ordering of the root links is based on their previous operating history. The fewer the errors associated with a particular root link, the higher its priority in the ordering. The root link with the best record is tried first. If the first attempt to connect a particular root node is not successful, the process is repeated a second time. The second try is used as a filter for transient faults.

The first step in setting up a root link is to configure the root node so that the port adjacent to the IOS is enabled and all its other ports are disabled. The second step is to verify that the hardware involved in the root link is operating properly and that this root node can be used as a springboard to the rest of the network. The absence of communication or protocol errors in the chain which sends the configuration command and receives the node's reply is evidence of a properly functioning communication link between the IOS and the root node. An optional set of diagnostic tests may be conducted at this point. These are described later in more detail. If the root node passes all the diagnostic tests or if the tests are bypassed, a determination is made about the ability of the root node to function as a jumping off point for the addition of the remaining nodes in the network. If diagnostic tests are performed, this determination is made by identifying a non-failed port on the root node which is adjacent to another node. However, when diagnostic testing is bypassed, this is accomplished by finding a link to an adjacent node which can be enabled without errors.

The algorithm for adding nodes to the network is designed to conduct an exhaustive search for a properly functioning connection to every node in the network. The failure of a single port of a node does not cause the entire node to be considered failed. However, some nodes may not be reachable by any path; the identity of these unreachable nodes is apparent only after this phase of the growth process is complete.

This stage of network growth begins after a root link has been established. The root node becomes the first entry in the spawning queue, a data structure used to control the growth of the network. An entry in the queue consists simply of a node which has been successfully added to the network but from which growth has not yet taken place. Two positions are marked in the queue: the top and the next entry. The top holds the node in the queue from which growth is currently taking place. This node is called the spawning node. The next entry is the next empty position in the queue. As nodes are added to the network, they are placed on the spawning queue at the next entry point and the next entry point is advanced to point to an empty position in the queue. As growth of the network proceeds, the topmost node in the spawning queue is removed from the queue and used as the jumping off point, or spawning node, for further growth. The root node becomes the first spawning node. Each node in the spawning queue is processed in turn until the queue is empty.

The processing of the spawning node proceeds on a port by port basis. The action taken depends on the kind of element found adjacent to each port. If the adjacent element is a remote GPC or a DIU, the spawning node and the port of the spawning node facing that element is recorded for future reference. These ports will be enabled after the network nodal growth is complete. However, if the adjacent element is a node whose status is idle, i.e. not yet part of the active tree, an attempt is made to enable the link to that node, referred to as the target node. If the attempt to enable the link between these nodes is not successful, the link is disconnected. If the attempt is successful, an optional set of diagnostic tests may be performed on the newly added node. If the tests are not performed, the target node is placed at the end of the spawning queue; otherwise, the target node is placed on the spawning queue only after it passes the diagnostic tests. When all the ports of the spawning node have been processed in this way, the next node in the spawning queue becomes the spawning node.

Figure 12 shows the entries made to the spawning queue for the growth of a fault free, six node network. Node 1, the root node, is the first entry. The three nodes adjacent to Node 1 are each added in turn to the network. As each node passes the set of diagnostic tests described below, it is added to the spawning queue. When all the nodes adjacent to Node 1 have been added to the network, Node 2 becomes the spawning node. Node 2 has one active link, an idle link adjacent to Node 3 and an idle link adjacent to Node 4. Since Node 3 is already active, the only node to be added to the network from Node 2 is Node 4. The next spawning node is Node 6. Node 5, the only idle node adjacent to Node 6, is the last node added to the network. Nodes 3, 4 and 5 each become a spawning node. However, since none of these nodes is adjacent to an idle node, no further nodes are added to the network or to the spawning queue which is now empty.

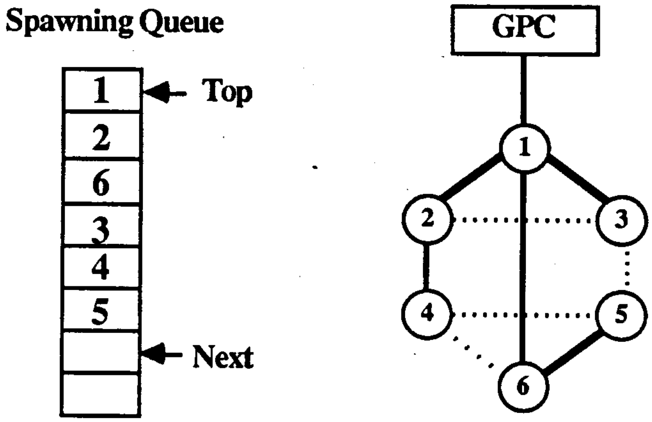


Figure 12. No Fault Growth Algorithm

The growth algorithm also detects and isolates babbling network components, thus making it a useful backup tool for network maintenance. When a port of a spawning node adjacent to a babbling port on the target node is enabled, the babbler is detected because its babbling transmissions interfere with the status report the spawning node sends following its reconfiguration. Following the detection of the babbler, the spawning node is sent another command instructing it to disable the port adjacent to the babbler, thus isolating the babbler from the rest of the properly functioning network. The method works because the network links are full duplex in the sense that separate physical data links exist for the transmission and reception of data. The reconfiguration command reaches the spawning node through a path not corrupted by the babbler. If the spawning node itself is babbling from the spawning port, the target node will not respond to the corrupted message. Thus the target node will not be connected to the babbler.

The use of the growth algorithm to isolate a node which is babbling from all its ports is illustrated in Figure 13. Node 2 is shaded to denote it as the babbler. When Node 1 is the spawning node, the attempt to connect Node 2 fails because the babbler causes violations of the established communication protocols. Hence, Node 2 is not added to the spawning queue. Nevertheless, Nodes 6 and 3 are added as before. Node 6 is the second spawning node from which Nodes 4 and 5 are added to the active tree. When Node 3 becomes the spawning node, a second attempt is made to reach Node 2. (A node may be babbling from one port only.) When this attempt fails, Node 4 becomes the spawning node. Since Node 2 is still not in the active tree, a third and final attempt to reach Node 2 is made from Node 4. Although Node 2 is babbling, the ports facing it on Nodes 1, 3, and 4 are disabled and therefore its faulty transmissions cannot disturb other network communication.

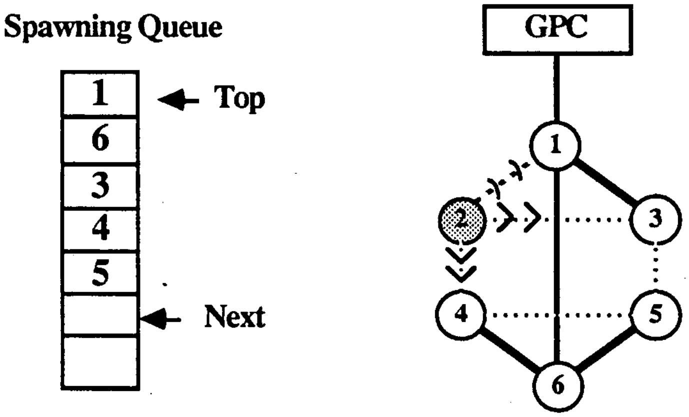


Figure 13. Network Growth Used To Isolate A Babbling Node

As each node is added to the network, a series of fault detection diagnostic tests may be performed. The tests are sequential in nature, and if any test fails, the remaining tests in the sequence are not performed. The first three tests are conducted on each port of the newly added node which is adjacent to an idle port of an idle node. This test sequence causes every network link to be exercised during the growth process.

The first test determines if the link between two nodes can be activated. It is performed by enabling the link between the newly added node and an adjacent node. If the attempt to enable the link is successful, the link is left in the enabled state so that the next test can be executed. If the link is not enabled, the ports on either end of the link are failed.

The second test determines whether or not the adjacent node transmits on a port after that it has been disabled. In this test, a configuration command is sent to the adjacent node over the newly enabled link instructing that node to disable all its ports. The node protocol is such that it carries out this command before transmitting a reply. A properly functioning node transmits a reply from all enabled ports to every command it receives. Since no ports are enabled, this message should not be transmitted. Thus, the node passes this test if no reply to the command is received. A node from which a reply is received is considered failed and has its status marked accordingly. When starting the third test, the adjacent node has all its ports disabled.

The third test determines whether or not the newly added node itself retransmits a message from a disabled port. This test requires three transactions to be transmitted on the network. The first transaction is sent to the newly added node commanding it to disable all of its ports except the inboard port connecting it to the established network. The second transaction is sent to the adjacent node commanding it to enable the port facing the newly added node for one transmission only. The third transaction is sent to the newly added node asking for its status. If the newly added node is functioning properly, it will not retransmit any messages, including the command making up the second transaction, to the adjacent node. On the other hand, if it has failed such that it does retransmit a message from a disabled port, the adjacent node will send a reply which may or may not be transmitted by the node under test back to the IOS. In either case, the transmission of this message causes the valid message detector for the port facing the adjacent node to record the transmission and to return this information as part of its status message. The newly added node passes this third test if no message from the adjacent node is received and the status indicator for the port in question shows no valid message received on that port. However, if it fails the test, the status of the node is marked failed. When the above three tests have been performed for every idle port of the newly added node, the newly added node remains configured such that only its inboard port is enabled. It is then ready for the last test.

If the preceding tests are completed without error, the last test is performed. This final test determines if the newly added node responds to commands sent to other nodes in the network. In this test, each node in the network is commanded to report its status, whether or not it is in the active tree. If an unconnected node responds to this command, it implies that the most recently connected node is responding to this address. Because of this protocol violation, this node must be disconnected from the active tree. Furthermore, its status is marked failed, since the address decoding function of a node is a central function, independent of the port receiving the address. A previously connected node could also respond with errors. This means that either this node has recently failed or the most recently added node is talking out of turn. This last added node is then removed from the network as described above. The node or nodes which had errors on the previous test are again queried for status. If the error indicators are gone, it confirms the talker out of turn hypothesis, and the status of the removed node is set to failed. If not, it indicates that a failure has occurred during the growth process. In the former case, the growth process is continued. In the latter case, the growth process must begin again from the start.

After every non-failed node in the network has been connected to the active tree, attempts are made to establish spare root links. This is accomplished by enabling the inboard facing port of every root node whose status is active but which is not connected to the active IOS or to a failed IOS. Up to two tries are made to obtain an error free response from the root node in this configuration. Each newly enabled root link is tested by collecting status using that interface. The results of the attempt to set up this connection are used to update the status of the interface. Successfully enabled root links have their status set to available; demerits are scored against an offending root link.

If any DIUs are present on the network, ports adjacent to them are enabled next. These components are checked for protocols established for all functioning network components. If a protocol violation is detected, the connection to the subscriber is disabled.

For regional networks, the ports adjacent to remote GPCs are enabled last. Since a GPC which is facing a port which is not enabled will not detect any network activity, it may be attempting to use the network at the time the port is enabled. This could result in errors being detected in the node's reply to its configuration command. Thus errors in the node status which is returned after enabling the root node port of a GPC are ignored. To verify that the GPC is in fact not babbling, however, the Manager asks for status from that node in a chain executed with contention. If errors are detected by this chain, that port is disabled.

After the initial growth of the network is completed, the status collection chain is executed from the active root link. This is analyzed by looking for any discrepancies between the status of the nodes in the network as determined by the process and the real time status of those nodes. This is to confirm or disprove the assumption that failures did not occur in the network during growth. If the real time data indicates the presence of a babbler, a failed IOS, or failed nodes which the growth process reported as active, then a discrepancy exists between the real state of the network and its state as determined during network growth. It cannot be determined whether these failures occurred after growth of the network was completed or during the growth of the network. Nodes which fail during the growth process but after they have been added to the active tree do not have a failure attributed to them but may cause other nodes to appear failed. Thus, the network is regrown. If the second try is unsuccessful, an intermittent failure exists on the network. The present algorithm does not handle intermittent faults. Hence, the network is declared to be inactive and the Network Manager is stopped. It may be restarted after the cause of the problem has been investigated.

The final step in network growth is to deselect transactions from the status chain which would query failed nodes for status. These nodes are isolated from the network and therefore cannot correctly return a status message even when the node is repaired. Errors detected by the status chain also trigger network reconfiguration. Therefore, the transactions are deselected. If the node is repaired, the operator can command the Network

Manager to reconnect the node to the network. At that time, the status transaction is reselected.

2.4.2 Fault Analysis

The purpose of this process is to analyze the data provided by the IOS after executing the status collection chain in order to identify both the type of fault responsible for the errors and the faulty network element itself. Four analyses are performed: raw data analysis, transient analysis, node data analysis, and error analysis. Each is described in this section.

While carrying out its principal function of sending and receiving data, the IOS detects various error conditions on the network. The IOS imparts this information to the processor through several status registers and through a buffer of status information appended to the incoming data of every input transaction. Further status information is obtained by programming the IOS to copy the values of status registers to memory locations in the DPM as the chain progresses. This information is referred to as raw data. The IOS Specification in Appendix C provides details about the error detection capabilities of the IOS. By analyzing this information certain failure modes are identified. These failure modes are: an interface failure due to a failed FTP channel or failed IOS hardware, the presence of a babbler on the network, and the failure of individual nodes to correctly follow the communication protocol. The order in which this status information is processed is important since the presence of a failed FTP channel connected to the active root link, a babbler, or a failed IOS precludes the analysis of other error indicators.

Raw data analysis begins by verifying that the channel connected to the IOS conducting this chain has not failed during chain execution. There are two parts to this diagnostic procedure: a data exchange pattern test and a call to GPC FDIR. Since GPC FDIR is a periodic process, a small amount of time may elapse between the failure of a channel and its detection by FDIR. The data exchange pattern test is used to detect a failed channel which GPC FDIR has not yet uncovered. If the channel with the active root link has failed, non-failed channels will obtain invalid data from its IOS/DPM. This data should not be processed since it could result in erroneous conclusions about the network. Similarly, if the channel failed after the last check with GPC FDIR (before the chain data was loaded into the DPM) but the failed channel has been resynchronized by GPC FDIR, then the data exchange pattern test will show no errors but again the chain data should not be processed since it may be invalid. To prevent this situation from occurring, GPC FDIR does not report the status of a channel as okay until it has undergone a trial period in a resynchronized state. This period is much longer than the longest chain delay. This means that errors resulting from a channel which failed before voted data was written to the DPM and which is now functionally resynchronized are still correctly attributed to the failed channel. The way in which checks are performed on the status of the channel which interfaces to the active IOS creates a window of time during which it is possible to determine whether or not the channel has failed.

If no channel failures are detected, raw data analysis proceeds with the status information provided by the IOS. If the value of the Chain Status Register (CSR) indicates that the chain did not complete, a command is written to the Interface Command Register (ICR) to stop the IOS. The IOS can be programmed to automatically time out individual transactions but it does not provide an overall time out for an entire chain. This time out condition is detected when the a chain does not complete in its allotted maximum execution time. A check is then made of other error indicators to determine if an incoming babbler was detected or if the IOS has failed. The indicators that are examined are the Interface Status Register (ISR) which detects a stuck on high condition in the network, the contention state of the IOS and the possession default indicator in the CSR, the extent to which the chain did complete as indicated by the value of the solicited chain pointer, the extent to which the IOS correctly performed its byte count zeroing function when executing a receive input instruction, and the ability of the DPM to pass a read/write pattern test.

If a chain does complete, the status indicators in the CSR are cleared. Thus the analysis to determine whether or not a babbler is present in the network is performed on the final value of the CSR which is saved by the end of chain program prior to commanding the ICR to switch modes (the definition of chain completion is a switch from the solicited to the unsolicited mode of operation). The final value of the CSR is examined for an indication of data transmission on the network while an output instruction is being executed by this IOS, for indications that data was transmitted on the network during the polling sequence or that a polling sequence was detected during data transmission by this IOS. The ISR is also examined for the presence of a stuck on high condition. Any of these protocol violations are evidence of a babbler on the network. If any of these errors are detected, a read/write pattern test is performed on the DPM to ensure that the error is due to a babbler and not a failed DPM.

Finally, the status information from each node transaction is examined for error information as follows. The HDLC status, which is saved after the transmission of the command to the node, indicates whether or not any framing or overrun errors occurred during this transmission. If this error is detected, the IOS is considered failed. If the byte count kept by the IOS on the data returned by the node still has its initial (non-zero) value, the IOS is considered failed. This value should range from zero to fifteen. Fifteen is the correct byte count, zero indicates no response is received from this node and any value in between is an incomplete transmission from the node. The IOS when operating correctly will zero this byte count and then start to increment it as data is received from the node. When the initial value has not been written over by the IOS, it is assumed that the IOS is not operating correctly.

In the cases where the error is attributed to a failed IOS, no further error processing is performed. However, some errors are attributed not to the IOS but instead to the transaction itself, i.e. the node, whose status is being analyzed. Thus, if the byte count has any other value except the correct byte count of fifteen, the error is attributed to the transaction itself and not to the IOS. In particular, if the byte count is zero, then no response was received from this node. The status of each transaction is then examined for the presence of HDLC protocol errors, the transmission by the node of an incorrect number of residual bits and an invalid sumcheck appended to the message. Should any of these errors be detected, a read/write pattern test is performed on the DPM to be sure that the error is not attributable to a failed DPM memory. When a transaction has no errors scored against it, the data associated with that transaction is returned to the Network Manager. However, if the transaction has errors scored against it, no data from that transaction is returned. When all the transactions have been subjected to this error analysis, the raw data analysis is complete.

Transient analysis provides a coarse filter between transient and permanent faults in the network. Automatic retries of those chains which produce error symptoms weed out transients. However, distinguishing between a true transient fault which occurs one time but is not reproducible and an intermittent or recurring fault is a very difficult task which is not pursued in the present design. The retry filter reduces the likelihood of a transient fault being diagnosed as a permanent fault. To deal with intermittent faults a system of demerits could be employed. If a fault is detected, but does not reappear in the retry, a demerit could be charged to the hardware causing the error if that can be determined or to the entire network if a more specific cause cannot be found. As spare link cycling proceeds, active links are retired temporarily from service. Over time the demerits will accumulate against a link (or set of links) experiencing intermittent faults. These links can be kept out of service for longer periods of time if their presence results in errors being detected. The demerit scheme has been implemented for root link errors. In the present design this information is used to prioritize root links for use in network regrowth and reconfiguration operations. Another area requiring further examination is the amount of time to wait between retries. The present model assumes faults occur more or less instantaneously. However, this model may not be accurate for faults due to damage events.

Data analysis is the process whereby the node status information is examined to detect faults in the network. A complete description of the data returned by a node is contained in Appendix D, the Node Specification. In particular, this analysis identifies a node which is transmitting from a port which should be disabled. This fault may or may not produce other error symptoms. Since valid data from the nodes is not included in the report if an interface failure or a babbler is present, data analysis only proceeds when these failure modes are not detected. Similarly, data is not included from nodes which have errors attributed to them during the execution of the status collection chain. However, since the failure mode detected by data analysis can produce error symptoms, data from error free node responses is analyzed even when other node responses do have errors. The reception of a valid message is recorded by an adjacent node even though it is configured to be disabled. (Adjacent ports are always in the same configuration, either both enabled or both disabled. They also have the same status, either both active, both idle or both failed.)

Thus, if a non-failed, disabled port reports the reception of a valid message, the node adjacent to that port is transmitting from a disabled port. Of course, several nodes may detect this fault if the node is transmitting from more than one disabled port but the fault is attributed to the node transmitting on the disabled port not the node receiving the faulty transmission. However, if more than one node is found to have this fault, the analysis is not successful.

As its name implies, error analysis is the process of deducing which network element produced the set of errors attributed to network nodes. Of course not all sets of errors are amenable to analysis. The input space of this subprogram has many combinations which do not pinpoint a specific network component as being faulty. Furthermore, the assumption underlying all the deductive reasoning in the error analysis, is that only one component has failed and this failure gives rise to all the error symptoms.

If all the nodes in the network have errors, error analysis attributes the errors to a root link failure. If some nodes have errors and some nodes do not, two possible failure modes are considered: a failed link (or node) through which no transmission takes place or a single node failure. The single node failure symptom could be indicative of a node which does not respond to commands but which continues to retransmit messages as it did before the failure. It could also be a node which itself is not failed but to whose address another node in the network responds. The single node failure is easy to diagnose since exactly one node in the status collection chain shows an error.

If more than one node has errors but fewer than all nodes have errors, the remaining problem is to determine if the cause of those errors is a link or node whose transmission/retransmission function is no longer operational. The basic idea is that when a link or a node fails in this way, then all nodes downline of this fault also have errors. The signature of such a failure is that nodes involved form a treelike pattern in the network. It should be noted that another failure mode which would produce a similar pattern of errors is a node which babbles on all its outbound ports. To determine if the observed errors fit the pattern for a failed link, node or outbound babbler is a three step process. The first step is to identify a node which qualifies as the root of the failed tree. Such a node is a node which had errors itself but which has an inboard port (a port which receives commands sent by the IOS) adjacent to a non-failed node. To prove this hypothesis, exactly one node should have this characteristic. If more than one such node exist, the fault is considered undiagnosable. However, if a root is found, the next step is to determine whether or not all nodes downline of the root had errors attributed to them. This is accomplished by a recursive algorithm. The algorithm processes information about the current node. The first value of the current node is the root of the failed tree as already determined. The nodes adjacent to the outboard ports of (i.e. downline of) the current node are examined. If such a node does not have errors attributed to it, the desired pattern is not present and the fault is considered undiagnosable. However, when the nodes downline of the current node do have errors, the recursion continues until every node downline of the root of the tree has been visited. If a treelike pattern is established, the last part of the pattern checking process can proceed. This step verifies that all the nodes which had errors appeared in the failed tree, i.e. no nodes with errors lie outside the tree. If nodes with errors are found outside the tree, the fault is considered undiagnosable. The final determination of whether or not the fault is due to a failed link, a failed node, or an outbound babbler is made during network reconfiguration.

Figure 14 shows a network which has a broken link. In this situation, the status collection chain would report no responses from the shaded nodes in the figure. Since Node 2 is the only node with an inboard port facing a non-failed node, it is identified as the root of the failed tree. Furthermore, all nodes downline of Node 2 are failed and no nodes outside the tree had errors. Thus, error analysis identifies this fault as a failed link between Node 1 and Node 2 or a failure of Node 2. The final identification of the fault takes place during network reconfiguration.

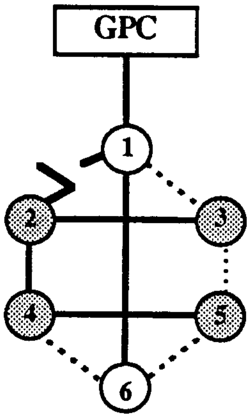


Figure 14. Identifying A Failed Link

2.4.3 Reconfiguration

The purpose of this process is to reconfigure the network so as to restore error free communication to all reachable, non-failed nodes in the network. The reconfiguration action depends on the type of failure determined by the fault analysis process. The fault identified in this report is actually a hypothesis about what is causing the errors on the network. The reconfiguration process, in effect, tests this hypothesis and then verifies that the network is again fully operational. Therefore, the network may go through several intermediate configurations before the reconfiguration process is complete.

The network fault analysis process identifies six classes of faults: a root link failure, a babbler, a link or node failure, a node which transmits from a disabled port, a single node failure, and an undiagnosable failure. A separate strategy exists to deal with each of these fault classes.

The reconfiguration process is considered complete when the node status chain is executed on the reconfigured network and does not detect any errors. The backup strategem for dealing with error phenomena which occur during a reconfiguration attempt but which are not anticipated is network regrowth. This is also the strategy when the fault analysis has not been able to diagnose the failure mode. In both cases the network is regrown without the fast grow option (i.e. growth without diagnostic testing) since the diagnostic tests uncover failure modes which may produce unanalyzable error patterns, such as nodes which respond to the addresses of other nodes and nodes which respond late and double fault occurrences.

In general, reconfiguration strategies are designed to deal with both active and passive faults in the hardware. Passive faults are characterized by the non-retransmission of data, i.e. a barrier or obstacle to data flow in the network. A disconnected cable is an example of such a fault; data cannot be retransmitted over this cable but transmission between other connections in the network is not affected. Active faults are characterized by the disruption of data flow in the network beyond the boundaries of the failed component itself. An IOS with a transmitter stuck on high is an example of this type of fault; the stuck on condition is retransmitted throughout the network, masking or disrupting transmissions between network connections remote from the failed part. Since different faults can produce identical error symptoms, e.g. a broken root link and an IOS transmitter stuck on high result in zero byte counts for all node responses, the reconfiguration algorithm must identify the specific cause of the problem so as to effect a repair. In this example, the passive fault (broken root link) only requires the selection of another root link. In the case of an active fault (the IOS stuck on high), the faulty component must also be isolated.

When the fault hypothesis is a failed root link, the hardware comprising the root link comes under scrutiny. This includes the IOS, the root node and the connecting cable. The interface status of the channel connected to the failed root link is updated to reflect the cause of the failure, a failed IOS or a failed channel. A spare root link, i.e. one that is connected to an operating FTP channel and IOS, is then chosen to establish a new FTP connection to the network. If no spare root links are found but at least one root link is connected to a failed FTP channel, this process suspends itself waiting for the outcome of GPC FDIR attempts to bring the failed channel back online. As a last resort, if no channels are brought back after a reasonable delay, an attempt is made to regrow the network. This causes failed root links to be tried one more time. However, in the case where spare root links are available, each is tried in turn until a non-failed root link is found or until another type of fault is diagnosed. The new interface is then used to execute the node status collection chain. Next, the node configuration table is updated to reflect the new root link; some outboard ports will become inboard and vice versa. Using this new configuration, an error analysis of the results of the node status chain is performed. If the analysis is unsuccessful, indicating the presence of an undiagnosable failure mode, the network is regrown. If no errors are detected, as would be expected in the case of a passive failure involving only the IOS, the inboard port of the root node, and the cable between them, the reconfiguration process is complete and the status of the new root link is marked active. If another root link failure occurs, the status of this root link is marked failed and the next available root link is tried. Finally, if either a babbler, or a link or node failure is detected, the root link switch is considered successful but the reconfiguration process is not yet complete. In this case, the reconfiguration process starts over again from the beginning, with a new root link but this time dealing with a new fault. This behavior would be expected for an active fault such as a babbling IOS or a passively failed root node which now must be removed from the network so that service to nodes downline of it can be restored.

When a babbler is detected in the network, the network is regrown using the fast grow option. A babbler is an active fault and includes a stuck on high condition detected by the IOS at its receiving interface to the network. (The IOS cannot observe a stuck on high condition on its transmitting interface.) For a network of N nodes which has a babbler, the cost of regrowing the network is N + P chains, where P is the number of spare ports on the babbling node which must be tried until a non-faulty one is found. Strategies to reduce this cost are possible in networks which are either maximally branching or fully linear. In the latter case a binary search could be used. The node in the middle of the bus would have its outboard port disabled and the location of the babbler deduced from the continued presence or the absence of babbler symptoms on the network after the reconfiguration. For the maximally branching network, a similar search could be conducted on each outboard branch of a node. If disabling the port which leads to a branch eliminates the babbler's symptoms, then the port is the gateway to the branch of the network containing the babbler. However, the network configuration may be a mix of these two basic patterns. The cost of finding the babbler is then not only a function of the number of chains necessary to identify and isolate the babbler, but also the cost of deciding which type of search to employ. In either case, once the babbler is identified by the search process, it must be isolated from the other nodes in the network. The decision as to which algorithm is least expensive depends to a large degree on the number of nodes in the network. More analysis of the problem is needed to make an informed choice as to which strategy should be used for a given network. The present design uses regrowth to reconfigure the network in which a babbler is present.

A failed node generates the same error pattern as a failed link. Thus, when the fault analysis reveals the presence of this failure mode, the reconfiguration algorithm must determine which fault has actually occurred and reconfigure the network accordingly. It is first assumed that a link has failed. The failed link is disconnected and an attempt to reach the failed node, i.e. the node immediately downline from the link, is made by using any spare ports on that node which are adjacent to nodes not in the failed node list. When this strategy fails to restore communication with the failed node (possibly because no spare ports are available), data is assembled which will allow each branch of the failed tree to be reconnected to the active network. This data consists of a list of nodes for each branch of the failed node, i.e. a separate list for each set of nodes which lie downline of each of its outboard ports. Only one successful connection to any spare port on a branch needs to be made in order to restore communication to the entire branch (and possibly to the failed node and all other nodes in the failed tree). A three transaction chain is used to reconnect the branch to the network. The first two transactions enable the ports on either side of the new link while the third transaction disables the former inboard port of the failed node in case the node adjacent to that inboard port is a babbler. If the failed node correctly returns its status, the repair is complete and the absence of errors is verified by collecting status from every node in the network. If the failed node is still not reachable, the port connecting this node to the present branch is disconnected and the proper functioning of the newly enabled link is verified. Then all the nodes on this branch are removed from the failed node set. The net effect of this process is to restore communication with all reachable nodes in the network while isolating the failed node. As communication to each branch is restored, the possible pool of spare links increases. Thus if any branch was not connected because of a lack of spare links, this branch is retried whenever a connection to another branch is successful. Any nodes which are still unreachable at the end of this process are marked failed.

If a node retransmits valid data from a port which should be disabled, the node must be removed from the network. This failure mode is distinguished from a babbler which is always transmitting a random bit stream or is stuck on high. When a babbling port is identified, the adjacent port of the neighboring node is disabled. This neighboring node will not retransmit from its other enabled ports anything received by the disabled port. Furthermore, the node will ignore any random bit patterns it receives. However, if the neighboring node receives a request for status addressed to itself on a disabled port, it will transmit its status from all its enabled ports, even though it does not retransmit the initial request. If this failed node is not removed, each time the manager asks for status from the node adjacent to this port, it would receive two valid commands to report its status. Only one response is expected. Once the first response is received, another node will be commanded to report its status. The second response of the node may interfere with the reply of a node whose transaction is later in the chain, making it appear that this next node has failed to respond correctly to a command. Once the failed node has been removed from the network, status is collected from the remaining nodes to verify that in fact the fault has been identified and isolated. If errors are still detected in the network, a full regrowth, with a complete set of diagnostic tests, is performed.

Removing a node is a simple matter if the node is a leaf; only the link connecting it to the network needs to be disconnected. This is accomplished with one chain. Otherwise, the nodes downline from the failed node need to be reconnected to the network through alternate links. If the node to be removed is the current root node, a new root link is selected. The nodes downline of each outboard port of the failed node are added to a reconnection queue, one queue for each branch which will be isolated when this node is removed from the network. Each of these nodes is also added to a set of unreachable nodes. The link connecting the inboard port of the failed node to the network is then disabled. Next, an attempt is made to reestablish a connection to each isolated branch via a spare link to a node in the reconnection queue of that branch from a node which is still reachable, i.e. is not a member of the unreachable node set. Only one such connection needs to be made to restore communication to all the nodes in the branch. After the new connection is enabled, the link connecting the failed node to this branch is disconnected. As each branch is reconnected, the nodes in that branch are removed from the reconnection queue. If any branch is successfully reconnected, branches which were not connected during earlier attempts are tried again since more spare links become available as communication is restored to nodes in other branches. This algorithm, while isolating the failed node, restores communication to every reachable node in the network. Nodes which cannot be reached because earlier failures have depleted the pool of spare links are marked failed.

Figure 15 illustrates the steps needed to isolate a node from the network. Suppose that Node 2 is to be removed from the network. First the link connecting Node 2 to Node 1 is disabled. When this step is completed, Nodes 2, 3, 4, 5, and 6 are also isolated from the GPC as shown in part II. Node 2 is the root of a tree with two branches, each of which must be reconnected in turn. By enabling the link between Nodes 1 and 6 and disconnecting the link between Nodes 2 and 4, one of these branches is reconnected to the active network as shown in part III. Finally, a link is enabled between Nodes 5 and 6 and the link between Nodes 2 and 3 is disabled. In this reconfiguration, Node 2 is isolated while preserving several links in the network. In larger networks, the performance gain of this approach over regrowth of the entire network is significant.

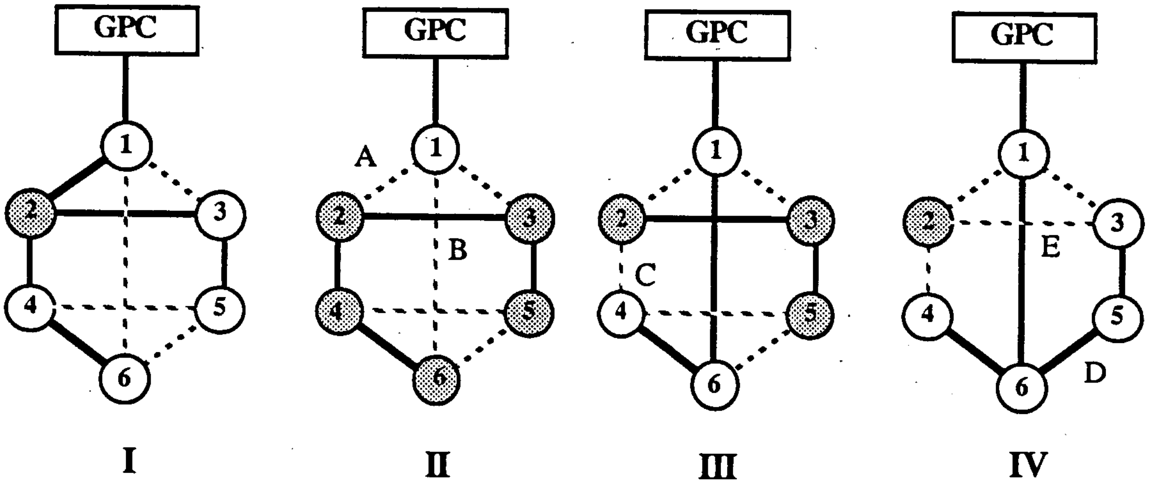


Figure 15. Removing A Node And Reconnecting Its Branches

A single node failure can occur if the failed node is a leaf node, if its retransmission function still works correctly but its status reporting capability is impaired, or if another node is responding to this node's address, making it appear that this node is failed. If the failed node is the current root node, before proceeding, a new root link is selected from the available spares, status is collected using this new root link, and a new error analysis is performed. The failed node is then isolated from the network, as described in the previous discussion, however, care is taken not to address this node directly because of the possible addressing problem. When the node is isolated, this node is again queried for its status. If a valid response is received, indicating the presence of a node which responds to the addresses of other nodes, the network is regrown with a full set of diagnostic tests to isolate this faulty node. Otherwise, an attempt is made to find an alternate route to this node using any port except its previously failed inboard port. The configuration command sent to this node as part of the link enabling procedure will disable the failed inboard port.

If two attempts to reconfigure the network have not succeeded in eliminating errors, then a full regrowth is called for. This is the back up reconfiguration strategy, used when all else fails.

3.0 I/O NETWORK MANAGEMENT SOFTWARE SPECIFICATIONS

3.1 I/O Network Manager

Process Name:	I/O Network Manager
Inputs:	I/O Network Identifier Network Topology Results of Spare Link Chain
Outputs:	Network Status Active Root Link Unreachable DIUs Network Usability Data for Spare Link Cycling
Requirements Reference:	I/O Network Functional Requirements, Section 2.2, 2.3
Notes:	None.

Description:

An instance of this process will be created for each network connected to a GPC. However, the process will remain in a quiescent state until it is activated by the Resource Allocator. At any given time, only one of the GPCs connected to an I/O network will host the activated Network Manager of that network. The activation of a Network Manager by the Resource Allocator will only require the scheduling of an existing process on the system. Memory allocation, process instantiation, and the initialization of variables used by this process will already have taken place. In this way, the activation of a Network Manager can be accomplished very quickly in real time if necessary.

When a Network Manager process is created, some software initializations take place which need to be performed by this process one time only. These include obtaining the I/O Network Identifier of the network it will manage, reading the Network Topology from the I/O Local Database, and obtaining an initial copy of Network Status from I/O Network Status. This sequence is accomplished during the power on phase of system operation. This preliminary work is not considered part of the routine operations of the Network Manager.

Once it has been activated, the Network Manager will grow or initialize its assigned network using the subprograms described in section 3.1.1. Once the network is established, it writes the status of the network hardware as determined in the growth process to Network Status where it can be examined by other system software services. It also generates the first set of commands to be used as Data for Spare Link Cycling. It signals the I/O Communication Manager that the network is ready for use by writing a value of In Service to Network Usability and again enters a quiescent state. At this point it can be deactivated and returned to its initially passive condition by a call from the Resource Allocator or it can be called upon to reconfigure the network or process the Results of the Spare Link Chain by the I/O Communications Manager. The subprograms involved in network maintenance are described in section 3.1.2. Each time the Network Manager initializes or reconfigures a network in response to a call from the I/O Communications Manager, it indicates that it has completed its actions on the network by writing a value of repaired to Network Usability. It also marks the Active Root Link and the list of Unreachable DIUs for use by the I/O Communications Manager. If the Network Manager is deactivated by the Resource Allocator it sets the Network Usability to out of service, thereby signaling the I/O Communication Manager that it is no longer managing the network.

The I/O Network Manager communicates with nodes by sending them commands and processing responses which are formatted in accordance with the specifications detailed in Appendix D, Node Specification. Section 3.1.3 describes the subprograms used to format these messages and give the details of some low level utility routines used by the Network Manager in network growth and maintenance operations.

3.1.1 Process Name: Network Growth

Inputs:

I/O Network Identifier
Network Topology
Root Links
Fast Grow Option
Current Channel
Network Status
Network Configuration
Root Link History

Outputs:

Current Channel
Network Status
Network Configuration
Root Link History

Requirements Reference: I/O Network Functional Requirements, Section 2.4.1

Notes: None

Description:

This process makes two attempts to grow the network specified in the Network Topology. The Network Topology describes all the interconnections which exist in the network on a node by node basis. Network growth is accomplished by a set of nested subprograms. The outermost subprogram verifies and validates the results of a second subprogram which assumes that, although hardware faults may be present in the network before the growth process commences, no additional faults will occur while growth is taking place. This second subprogram conducts the real business of network growth. It calls other subprograms to finish the work of growing the network and then returns a boolean parameter to its caller which indicates whether or not the network has been grown successfully. For growth of a network to be considered successful, an active root link must connect the GPC to the network (this implies a properly functioning IOS and, except in the trivial case of a one node network, a root node which is able to communicate with at least one adjacent node), and all non-failed nodes in the network must be part of the active tree. If the subprogram indicates that growth is not successful, it is called a second time after a short delay has expired. However, if the subprogram indicates that growth is successful, the calling process causes the status collection chain to be executed from the active root link. It analyzes this data by looking for any discrepancies between the status of the nodes in the network as reported by the subprogram and the real time status of those nodes. This is to confirm or disprove the assumption made by the inner growth subprogram that failures did not occur in the network during growth. The validity of the assumption is somewhat dependent on the length of time it takes to grow the network. The faster growth is completed, the less likely a failure will occur during growth. Thus this test makes the growth process more robust. If the real time data indicates the presence of a babbler, a failed IOS, or failed nodes which the growth process reported as active, then a discrepancy exists between the real state of the network and its state as recorded by the subprogram. It can not be determined whether these failures occurred after growth of the network was completed or during the growth of the network. Nodes which fail during the growth process but after they have been added to the active tree do not have a failure attributed to them but may cause other nodes to appear failed. Thus, if a discrepancy exists, the network is regrown. If the second try is unsuccessful, a serious problem exists on the network requiring either a function migration or operator intervention to correct the problem. In the second case, the network is declared to be inactive and the Network Manager is stopped. It may be restarted after the cause of the problem has been investigated. The choice of action is made by the Resource Allocator. The rest of this section describes the algorithm used by the second subprogram to grow the network. Subsequent sections describe parts of this process in greater detail.

The inner subprogram which assumes no faults occur in the network after growth has begun has access to the same input and output parameters as the calling process. However, faults can occur during network growth. If these faults are detected, the growth process begins again from the top. An example of this type of fault is the failure of the active IOS. This subprogram tries up to two times to grow the network. If faults continue to occur during the growth process or if an active root link as defined above is not found, the subprogram indicates to the calling process that growth was not successful.

Prior to growing the network, some software initialization is necessary. The status of each node and network interface in Network Status is set to idle. The growth algorithm assigns some non-idle value to the status of each network interface before completing its work. The status of each port is also set to idle, however, idle ports in non-failed nodes are acceptable and the port status will not necessarily have a non-idle value after growth is complete. The status of the hardware components in the network reflect their current status only, based on the most recent data about those components, not on their history of failure. Another part of initialization concerns the node status collection chain which resides in Dual Ported Memory. It is modified to ensure the execution of a status collection transaction for every node in the network. Finally, the counters which keep track of the number of DIUs and interfaces to remote GPCs are initialized to zero.

The growth of the network begins by establishing an active root link to one of the root nodes and ensuring that this root node has a port which can be used as the springboard to the rest of the nodes in the network. If an active root link is found, the remaining nodes are added to the active tree. The algorithm for adding nodes to the network will be discussed in section 3.1.1.2. This process conducts an exhaustive search for a properly functioning connection to every node in the network. Once this connection is established, the status of the node is upgraded to active. The failure of a single port of a node does not cause the entire node to be considered failed. Some nodes may not be reachable by any path. However, the identity of these unreachable nodes will be apparent only after this phase of the growth process is complete. Since any node whose status is idle after network growth is completed is not reachable by any port, its status will be given a value of failed.

After the nodal network is established through the active root link, the spare root links to the network must be enabled and tested. If any DIUs are present on the network, nodal ports adjacent to them are enabled next in a process called adding DIUs. Finally, if this is a regional network, ports adjacent to remote GPCs must be enabled in a process called adding remote GPCs.

The growth process is summarized below. Further details on each aspect of the process are available in the indicated sections.

```
Repeat until growth is successful or two attempts fail to produce a stable
network
    Establish a working connection to a root node (3.1.1.1)
    If an active root link is established then
        Add remaining nodes to the network (3.1.1.2)
        Mark idle nodes failed
        Add spare root links (3.1.1.4)
        Add DIUs (3.1,1.5)
        Add Remote GPCs (3.1.1.6)
        Collect Node Status from all nodes in network as defined by topology
            (3.1.2.1)
        Validate Network Status
        If no discrepancies in Network Status then
            Network is grown successfully
```

3.1.1.1 Process Name: Establish Root Link

Inputs:

I/O Network Identifier
Network Topology
Root Links
Fast Grow Option
Current Channel
Network Status
Network Configuration
Root Link History

Outputs:

Current Channel
Network Status
Network Configuration
Root Link History
Spawning Queue
Active Root Node Flag

Requirements Reference: I/O Network Functional Requirements, Section 2.4.1

Notes: None

Description:

This process is called as the first step in the growth of a network. Its job is to set up a properly functioning a connection to the network. The hardware involved in the connection consists of an IOS, a root node, and the link between them. Establishing the connection is a two step procedure. It requires that this hardware be put in a state which supports communication between the GPC and the root node and that the correct operation of this hardware be verified.

Since networks in general may have more than one physical connection to a GPC, the approach taken is to order the root links in some way and then to try them in turn until a properly functioning connection is found. The ordering of the root links is based on their previous operating history. The fewer the errors associated with a particular root link, the higher its priority in the ordering. The tally of errors is kept in the Root Link History. The root link with the best record is tried first. The process is complete when a properly functioning root link is found.

Each root link is tried in turn until a fully functional connection is established. If the first attempt to connect a particular root node is not successful, the process is repeated a second time. The second try is used as a filter for transient faults. When a successful connection is made, the value of the Root Link Active Flag is set to true to indicate to the calling subprogram that a working connection to the network has been made. Additionally, the Current Channel is given the value of the FTP channel to which the active IOS is connected and the Spawning Queue is initialized with the root node. If, on the other hand, no root link is established, the value of the Root Link Active Flag is set to false.

The first step in setting up a root link is to configure the root node so that the port adjacent to the IOS is enabled and all its other ports are disabled. This is accomplished by preparing a command to the root node which causes it to configure its ports as described and then executing a chain which sends the command to the node. Setting up the command is accomplished by means of a utility routine described in Section 3.1.3. Sending the command is accomplished by means of another utility described in Section 3.2.4.1. The second utility returns a Configuration Report describing the errors, if any, that were detected during the execution of the chain.

The second step in setting up the root link is to verify that the communication hardware is operating properly and that this root node can be used as a springboard to the rest of the network. First the information in the Configuration Report is analyzed to determine whether or not error free communication with the root node has taken place. The absence of errors is evidence of a properly functioning full duplex communication link and implies that the IOS and node hardware is fully operational. If the Fast Grow Option has not been selected, a full set of diagnostic tests are conducted on the root node. These are described in more detail in Section 3.1.1.3. If the root node passes all the diagnostic tests or if the tests are bypassed because the Fast Grow Option is chosen, a determination is made about the ability of the root node to function as a jumping off point for the addition of the remaining nodes in the network. If diagnostic tests are performed, this determination is made by identifying a non-failed port on the root node which is adjacent to another node. This can be done by using the Network Topology and the Node Status information. However, in the case when diagnostic testing is bypassed, this is accomplished by finding a link to an adjacent node which can be enabled to support full duplex communication. Once a root node which has at least one port which can be enabled to communicate with an adjacent node has been found, the Node Status of this node is marked active, the port status of the port adjacent to the IOS is also marked active, the Interface Status of this interface is marked active, and the configuration of this node is recorded in Network Configuration. To accomplish the latter means that the Node Configuration of this node shows the enabled port marked Inboard and the other ports marked Idleport. This process is then complete.

The preceding paragraph describes the actions taken by this process if the Configuration Report indicates that no protocol errors are detected when the command to change its port enable is sent to the root node. When errors are detected, they are processed before either a second try is made or the next root link is tried. The error processing proceeds as follows. If the error detected is a channel failure, a retry is not undertaken since it is unlikely that the channel can be restored in time to make this a viable root link. Instead, the Interface Status is marked Failed Channel and the next root link is tried. If the error detected is a Babbler, the Contention Option is set to run the configuration chain without contention before the retry is undertaken. A babbler on the network prevents a contention poll from running to completion. Therefore, the command to the root node to disable all but its inboard port will never be transmitted. However, the babbling may be due to some remote component. Since the links are full duplex, only the incoming transmission line may be affected. Executing the chain without contention causes the command to the root node to be transmitted over the unaffected outgoing transmission line. Once the root node receives and obeys this command, the babbling transmissions will not be retransmitted by the root node to the IOS and the babbler error indicators will not continue to register the presence of a babbler. If any other errors are detected such as no response or HDLC protocol errors, the same configuration chain is simply run a second time. If errors are detected on the second try, the interface is marked failed IOS, the status of the port adjacent to the IOS is also marked failed and the next root link is tried.

3.1.1.2 Process Name:	Adding Nodes to Network
Inputs:	I/O Network Identifier Network Topology Fast Grow Option Current Channel Network Status Network Configuration Spawning Queue
Outputs:	Network Status Network Configuration Network Subscribers
Requirements Reference:	I/O Network Functional Requirements, Section 2.4.1
Notes:	None

Description:

This growth algorithm generates the shortest path from the source processor to any node in the network. Furthermore, if a path exists to any node in a network, this algorithm ensures that it will be found and activated, even if the network is degraded by failures.

This subprogram is called into service after Establish Root Link (3.1.1.1) has established a fully operational root link to a root node of this network. The root node is the first entry in the spawning queue, a data structure used to control the growth of the network. An entry in the queue consists simply of the node number of a node which has been successfully added to the network but from which growth has not yet taken place. Two pointers are used to mark positions in the queue: the Top and the Next Entry. The Top points to the node in the queue from which growth is currently taking place. This node is called the spawning node. The Next Entry points to the next empty position in the queue. As nodes are added to the network, they are placed on the spawning queue at the Next Entry point and the Next Entry point is incremented to point to an empty position in the queue. The spawning queue thus grows from the bottom. As growth of the network proceeds, the topmost node in the spawning queue is removed from the queue and used as the jumping off point for further growth. The root node becomes the first spawning node. The growth algorithm then enters a loop in which each node in the spawning queue is processed in turn until the spawning queue is empty.

The processing of the spawning node proceeds on a port by port basis. The action taken depends on the kind of element found adjacent to each port. The identity of that element is obtained from the Network Topology. If the adjacent element is a remote GPC, the spawning node and the port of the spawning node facing that element is placed on the GPC subscriber list. If the adjacent element is a DIU, similar entries are made to the DIU subscriber list. These ports will be enabled after the network nodal growth is complete. However, if the adjacent element is a node whose status is idle, i.e. not yet part of the active tree, an attempt is made to set up a functional link to that node, referred to as the target node. If the attempt is successful, the target node is placed at the end of the spawning queue. Creating a functional link requires that a port of the spawning node and a port of the target node be enabled; the spawning node is enabled first. Enabling this link is accomplished by a utility subprogram, Enable Link, described in section 3.1.3. If the attempt to enable the link between these nodes is not successful, the Enable Link subprogram will disconnect the nodes. If the reason for the failure is the detection of a babbler, this subprogram runs a test for a babbler to ensure that the attempt to disconnect the babbling node was successful. If it is not, an exception is raised which causes the growth to begin again from the start. When the attempt to enable the link is not successful, the link is left in an disconnected state and the the status of the two ports used in the connection are marked failed. When the link is connected successfully and the fast growth option is selected, the target node is added to the spawning queue, its status is marked active, the status of the ports connecting the spawning node and the target node are marked active, and the new configuration of the two nodes is noted in Network Configuration. The latter requires the configuration of the port in the spawning node to be marked outboard and the configuration of the port in the target node to be marked inboard, reflecting the flow of data with respect to the active IOS. However, if the fast growth option is not selected, the target node is subjected to a set of diagnostic tests which it must pass before being added to the spawning queue. These tests are described in section 3.1.1.3. If it does not pass these tests, the status if the target node is marked failed. If it does pass the tests, however, it is added to the spawning queue and the various status records are updated as before. When all the ports of the spawning node have been processed in this way, the next node in the spawning queue becomes the spawning node. Network growth continues until the spawning queue is empty.

As mentioned above, this algorithm detects and isolates babbling network components, thus making it a useful backup tool for network maintenance. When a port of a spawning node adjacent to a babbler is enabled, the babbler is detected because its babbling transmissions interfere with the status report the spawning node sends following its reconfiguration. Following the detection of the babbler, the spawning node is sent another command instructing it to disable the port adjacent to the babbler, thus isolating the babbler from the rest of the properly functioning network. The method works because the network links are full duplex and the reconfiguration command will reach the spawning node through the data line not corrupted by the babbler. If the spawning node itself is babbling from a spawning port, the target node will not respond to the corrupted message. Thus the target node will not be connected to the babbler.

3.1.1.3 Process Name:	Diagnostic Testing
Inputs:	Node Under Test Inboard Port of Node Under Test I/O Network Identifier Network Topology Current Channel Network Status Network Configuration
Outputs:	Network Status Network Configuration Passed Diagnostic Tests
Requirements Reference:	I/O Network Functional Requirements, Section 2.4.1
Notes:	None

Description:

For each port of the Node Under Test adjacent to an idle node, a series of fault detecting diagnostic tests is performed. The tests are sequential in nature, and if any test fails, the remaining tests in the sequence are not performed. The first test determines if the link between two nodes can be activated. The second test determines whether or not the adjacent node transmits on the port adjacent to the Node Under Test after that port has been disabled. The third test determines whether or not the Node Under Test itself retransmits a message from a disabled port. After this set of tests is completed without error, the last test is performed. This final test determines if the Node Under Test talks out of turn to addresses of other nodes in the network. This test could be expanded to include the addresses of DIUs on this network.

The first test is performed by using the Enable Link Utility routine described in section 3.1.3. If the attempt to enable the link is successful, the link is left in the enabled state so that the next test can be executed.

In the second test, a configuration command is sent to the adjacent node over the newly enabled link instructing that node to disable all its ports. The node protocol is such that it will carry out this command before transmitting a reply. A properly functioning node transmits a reply from all enabled ports to every command it receives. Since no ports are enabled, this message should not be transmitted. Thus, the node passes this test if no reply to the command is received. A node from which a reply is received is considered failed and has its status marked accordingly. When starting the third test, the adjacent node has all its ports disabled.

In the third test, a chain of three transactions is transmitted on the network. The first transaction is sent to the node under test commanding it to disable all of its ports except the inboard port connecting it to the established network. The second transaction is sent to the adjacent node commanding it to enable the port facing the node under test for one transmission only. The third transaction is sent to the node under test asking for its status. If the node under test is functioning properly, it will not retransmit any messages, including the command making up the second transaction, to the adjacent node. On the other hand, if it is has failed such that it does retransmit a message from a disabled port, the adjacent node will send a reply which may or may not be transmitted by the node under test back to the IOS. In either case, the transmission of this message will cause the activity detector and the valid message detector for the port facing the adjacent node to record the transmission and to return this information as part of its status message. The node under test passes this third test if no message from the adjacent node is received and the status indicator for the port in question shows no activity and no valid message received. If the node under test fails this test, its status and the status of all its ports is marked failed. When the above three tests have been performed for every port of the node under test adjacent to an idle node, the node under test is configured so that only its inboard port is enabled. It is then ready for the last test.

In the last diagnostic test, each node in the network is commanded to report its status, whether or not it is in the active tree. If an unconnected node (i.e. one which is not on either the spawning queue or the active node list) responds to this command, the most recently connected node is talking out of turn to this address. This newly added node must be disconnected from the active tree by setting the correct spawning node port to a null state. Furthermore, its status in Node Status is marked failed, since the address decoding function of a node is a central function, independent of the port receiving the address. A previously connected node could also respond with errors. This means that either this node has recently failed or the most recently added node is talking out of turn. This last added node is then removed from the network as described above. The node or nodes which had errors on the previous test are again queried for status. If the error indicators are gone, it confirms the talker out of turn hypothesis, and the status of the removed node is set to failed. If not, it indicates that a failure has occurred during the growth process. In the former case, the growth process is continued. In the latter case, the growth process must begin again from the start.

3.1.1.4 Process Name:	Connecting Spare Root Links
Inputs:	I/O Network Identifier Network Topology Root Links Fast Grow Option Current Channel Network Status Network Configuration Root Link History
Outputs:	Network Status Network Configuration Root Link History
Requirements Reference:	I/O Network Functional Requirements, Section 2.4.1
Notes:	None

Description:

This subprogram attempts to enable the inboard facing port of every root node whose status is active but which is not connected to the active IOS or to a failed IOS. Up to two tries are made to obtain an error free response from the root node in this configuration. If the fast grow option is not selected, each newly enabled root link is tested by collecting status using that interface. The results of the attempt to set up this connection are used to update the status of the interface and root nodes. Successfully enabled root links have their status set to available. Errors are tallied against the offending root link in Root Link History.

If the first response to the command enabling the port adjacent to the IOS reveals the presence of a babbler, the second try is sent without contention. If this also indicates a babbler is present, the root link is disconnected and the interface is marked failed IOS.

3.1.1.5 Process Name:	Adding DIUs
Inputs:	I/O Network Identifier Network Topology Current Channel GPC Subscriber List Network Status Network Configuration
Outputs:	Network Status Network Configuration
Requirements Reference:	I/O Network Functional Requirements, Section 2.4.1
Notes:	None

Description:

The ports adjacent to DIU subscribers on the subscriber list are enabled one at a time. If no errors are reported from this transaction, the port remains enabled. However, if errors are reported the port is returned to an inactive state. An error detected after enabling this port is due to a babbling DIU or a failure in the node adjacent to the DIU which occurred after the node is successfully added to the network. In either case, at this point the only action taken in response to the detection of an error is to return the port to an idle state. The final configuration of the port is recorded in Network Configuration; the status is recorded in Network Status. If the connection is operational, the port is marked outboard and its status is marked active.

3.1.1.6 Process Name: Adding Remote GPCs

Inputs:	I/O Network Identifier Network Topology GPC Subscriber List Current Channel Network Status Network Configuration
Outputs:	Network Status Network Configuration
Requirements Reference:	I/O Network Functional Requirements, Section 2.4.1
Notes:	None

Description:

The ports adjacent to GPC subscribers on the subscriber list are enabled one at a time. Once the network manager gives other GPCs access to the network, the manager must use the contention rules which govern access to a multi-user network. Since a GPC which is facing a port which is not enabled will not detect any network activity, it may be attempting to use the network at the time the port is enabled. This could result in errors being detected in the node's reply to its configuration command. Therefore, errors in the node status which is returned after enabling the root node port of a GPC are ignored.To verify that the GPC is in fact not babbling, however, the manager must ask for a status read of that node with contention. If the transmission has errors, that port is returned to a null status. This command is first sent with contention. Only if the errors persist will the command be sent without contention. This phase of network growth is complete when all the ports on the subscriber list have been enabled and verified for proper functioning. Node Configuration and Node Status are updated following each verification transaction.

3.1.2 Process Name: Network Maintenance

Inputs: I/O Network Identifier, Network Topology, Root Links, Current Channel, Network Status, Network Configuration, Error Report, Root Link History

Outputs: Current Channel, Network Status, Network Configuration, Root Link History

Requirements Reference: I/O Network Functional Requirements, Sections 2.4.2, 2.4.3

Notes: None

Description:

The various services provided by this process are scheduled by the I/O Communication Manager. The services provided are: status collection from the nodes in the network, spare link cycling and fault identification and network reconfiguration. If spare bandwidth is available on a network, the I/O Communication Manager may choose to collect node status or to retire an active link and bring a spare link into service. If errors are detected during the execution of an I/O request for an application program, during the execution of the node status collection chain,.or during the attempt to cycle a spare link, the I/O Communication Manager takes that network out of service and allows the Network Manager to have sole access to the network until the reconfiguration has restored full service to all non-failed network nodes and subscribers.

3.1.2.1 Process Name: Network Status Collection

Inputs: I/O Network Identifier
Active Root Link
Logging Enable

Outputs: Status Collection Report

Requirements Reference: I/O Network Functional Requirements, Section 2.4.2

Notes: None.

Description:

Network Status Collection is the fault detection mechanism of the Network Manager. When this subprogram is called, it is assumed that the Network Manager is in control of the interface to the network, i.e. that the Communications Manager has taken the network out of service. In addition to collecting status from each non-failed node in the network, this subprogram performs some preliminary analysis of error information. This information is obtained by the IOS as it attempts to execute the status collection chain, a chain which is always executed with contention. The details of the execution of this chain are given in Section 3.2.4.2.

The Status Collection Report provides the Network Manager with a summary of the error information obtained from a preliminary analysis of the data which the IOS provides after executing the a status collection chain. When the IOS transmits messages on the network to a node, it observes aspects of the communication and records those observations in registers and buffers for later processing. This constitutes a first stage of fault detection and includes detection of the failure of a node to transmit a response to a command in a reasonable amount of time, the presence of transmission errors on the network during a response from a node, the incorrect number of words in a response, and other violations of the communication protocol. In addition to detecting errors on transactions to individual nodes, the overall performance of the network is monitored for failures which impede the proper functioning of the contention sequence. These failures include a babbler which is flooding the bus with meaningless signals and a data line which is holding the network in a "stuck on one" condition.

The summary presents a synopsis of the information provided by the IOS with conclusions drawn about the following error conditions: an interface failure, a babbler and individual errors detected for each node. If the summary reports that an interface failure has occurred, it also states whether the cause is a failed IOS or a failed channel connected to the active IOS. If the summary reports that a babbler is present on the network, it also specifies whether the babbler was detected during contention for the network or during data transmission. When either of these errors are present, no further data is provided since the integrity of this data is in question. Furthermore, the Network Manager's strategies for reconfiguring the network to eliminate these problems do not require information from individual nodes. Finally, if neither an interface failure or a babbler is detected, an error indicator is provided for each active node in the network. This error indicator simply notes that an error has occurred. The error could be due to a variety of causes, including a no response error, an HDLC protocol violation, or a check sum error. The type of error is logged in the I/O Network Error Log, however, it is not passed back to the Network Manager, since its logic does not require this level of granularity in order to correctly reconfigure the network.

In future implementations of the Network Manager, additional sources of information may be used as part of the status collection aspect of network FDIR. In particular, this could include information from the I/O Communications Manager about errors detected during transactions with specific DIUs. For regional I/O networks, I/O Communication Managers in remote GPC subscribers to the network would send this information to the Network

Manager over the intercomputer network. Errors reported by the I/O Communication Manager when no errors have manifested themselves during node status collection are evidence of transient faults in the network, faults with DIUs or connections to DIUs, or of other faults which the Network Manager's use of the network does not trigger. This information is useful in building up a statistical profile of network components. In future AIPS implementations it may be possible to support multiple links to DIUs from different nodes or even between different networks. With this capability, errors which result from faults in the link between a node and a DIU (or because a node connected to a DIU is failed) could result in a reconfiguration involving the active link to a DIU. Since the actual data flow in the network is a function of the GPC using the network, errors detected by remote GPCs may not be detectable by the Network Manager.It is possible to devise algorithms for isolating this type of fault. However, this work is beyond the scope of the present implementation.

3.1.2.2 Process Name:	Network Fault Analysis
Inputs:	I/O Network Identifier Active Root Link Network Topology Network Configuration Status Collection Report
Outputs:	Error Analysis Report
Requirements Reference:	I/O Network Functional Requirements, Section 2.4.2
Notes:	None.

Description:

The purpose of this process is to analyze the data provided by the Status Collection Report in order to identify the both the type of fault responsible for the errors and, if possible, the faulty network element itself. This is accomplished by a set of subprograms. There are three types of analysis which are performed: transient detection, data analysis, and error analysis. Each is described in this section.

The Network Manager first filters the data in the Status Collection Report through transient analysis. If this subprogram concludes that the error is a transient, no further analysis is performed. However, if the fault is permanent, the data is screeened for errors first by data analysis and second by error analysis. In some cases the analysis of the fault is not completed by these subprograms; additional information is necessary before a final conclusion can be drawn. In such cases, the analysis is continued by the subprograms which make up Network Reconfiguration. This analysis is discussed in detail in Section 3.1.2.3. That section also describes the actions taken by the Network Manager in response to the various conclusions arrived at in the network fault analysis process discussed here.

Transient Analysis is a subprogram which discriminates between transient and permanent faults in the network. It accepts a Status Collection Report as an input parameter. If this parameter indicates a fault is present in the network, it collects network status again and compares the second report with the first. If the second report finds no faults in the network, then the fault is assumed to be a transient. If the error reports are identical, then the fault is deemed permanent, i.e. not transient in nature. Finally, if both reports agree that a fault has occurred, but disagree on the nature of the fault, network status is collected a third time. In this case, if the second and third reports agree, the fault is judged to be permanent; otherwise a transient error is declared. The argument here is that perhaps a permanent fault occurred during the execution of the first status collection chain. If this is the case, the errors observed by the first and second status collections could be different. Consider the following scenario. During the first status collection, a node reports its status correctly, but then a link leading to that node is damaged, and other nodes downline of the first node have errors logged against them. When status is collected a second time, the node in question also has an error. Thus the first two reports do not agree, but the second and third reports do.

Of course this is a simplistic approach to a very difficult problem, namely differentiating between transient and intermittent faults. A more sophisticated approach would be to maintain a statistical history of faults and use this as the basis for isolating a component which is intermittently faulty. However, this analysis is deferred to more advanced implementations of network fault analysis.

Data analysis is the second major part of network fault analysis. Data analysis is the process whereby the status information returned by the nodes is reviewed for the purpose of extracting information about faults in the network. A description of the data returned by a node is contained in Appendix D, the Node Specification. These faults may or may not produce other error symptoms. This subprogram takes the Status Collection Report as an input parameter. Since valid data from the nodes is not included in the report if an interface failure or a babbler is present, data analysis only proceeds when these failure modes are not detected. Similarly, data is not included from nodes which have errors attributed to them during the execution of the status collection chain. However, since these failure modes can produce error symptoms, data from error free node responses is analyzed even when other node responses do have errors. The purpose of this analysis is to detect a node which is transmitting from a port which should be disabled. The transmission may be simple, random noise or a valid message retransmitted by a disabled port due to some fault in the node hardware. This is detected when a node records any activity on the network (i.e. a change in voltage from low to high or vice versa) or the reception of a valid transmission by a non-failed port which the Network Configuration shows to be disabled or idle.

(Adjacent ports are always in the same configuration, either both enabled or both disabled. They also have the same status, either both active, both idle or both failed.) If this condition was detected previously, the status of the port which is adjacent to the failed port will have a failed status, as well as disabled configuration. Hence, even though the port records the continued presence of the babbler, the correct reconfiguration has been made to contain the babbler and repeated error processing is neither necessary nor desirable. If more than one node is found to have this fault, a report indicating an unsuccessful analysis is returned by this subprogram. (Of course, several nodes may detect this fault but the fault is attributed to the node transmitting on the disabled port not the node receiving the faulty transmission.) When this fault is not present in the network, the report returned indicates no data errors were found. Finally, if a node is found with this fault, the error report indicates this fact along with the ID of the faulty node.

Error Analysis is the third and final subprogram in network fault analysis. As its name implies, error analysis is the process of deducing which network element produced the set of errors recorded in the Status Collection Report. Of course not all sets of errors are amenable to analysis. The input space of this subprogram has many combinations which do not pinpoint a specific network component as being faulty. In these cases, the subprogram returns a value of undiagnosable errors. Furthermore, the assumption underlying all the deductive reasoning in the error analysis, is that only one component has failed and this failure gives rise to all the error symptoms.

If the Status Collection Report indicates that an interface failure has occurred, the error analysis report attributes the errors to a root link failure, indicating the root link which failed and the cause of the failure, either failed IOS or failed FTP channel. In a similar manner, if a babbler is reported, the error analysis report attributes the errors to a babbler. If neither of these errors is present, the analysis proceeds with an examination of the errors attributed to non-failed nodes in the network.

If all the nodes in the network have errors, the error analysis report attributes the errors to a root link failure, indicating the root link which failed as before but also indicating the cause as a failed IOS. If some nodes have errors and some nodes do not, two possible failure modes are considered: a failed link (or node) through which no transmission takes place or a single node failure. The single node failure symptom could be indicative of a node which does not respond to commands but which continues to retransmit messages as it did before the failure. It could also be a node which itself is not failed but to whose address another node in the network responds. The single node failure is easy to diagnose since exactly one node in the Status Collection Report shows an error. The reconfiguration strategy used in this case is described in Section 3.1.2.3. If more than one node has errors but fewer than all nodes have errors, the remaining problem is to determine if the cause of those errors appears to be a link or node whose transmission/retransmission function is no longer operational. The basic idea is that when a link or a node fails in this way, then all nodes downline of this fault also have errors. The signature of such a failure is that nodes involved form a treelike pattern in the network. It should be noted that another failure mode which would produce a similar pattern of errors is a node which babbles on all its outbound ports. To determine if the observed errors fit this case is a three step process. The first step is to identify a node which qualifies as the root of the failed tree. Such a node is a node which had errors itself but which has an inboard port (a port which receives commands sent by the IOS) adjacent to a non-failed node. To prove this fault hypothesis valid, exactly one such node should have this characteristic. If more than one such node exists, the fault is considered undiagnosable. However, if a root is found, the next step is to determine whether or not all nodes downline of the root had errors attributed to them. This is accomplished by a recursive subprogram. The subprogram accepts a node as a parameter; the node is referred to as the current node. The first call to the subprogram passes the root of the failed tree as the input parameter. The subprogram examines the nodes adjacent to the outboard ports of the current node. If such a node does not have errors attributed to it, the subprogram returns a value of false and the fault is considered undiagnosable. However, if a treelike pattern is established, the last part of the pattern checking process can proceed. This step verifies that all the nodes which had errors appeared in the failed tree, i.e. no nodes with errors lie outside the tree. If nodes with errors are found outside the tree, the fault is considered undiagnosable. If all three steps in the process support the failed link/failed node hypothesis, an error analysis report is returned stating the fault is a failed link or a failed node. Additional information contained in the report is the node number of the failed root of the tree, the port number of the inboard port of this node, and a list of nodes in the tree. The final determination of whether or not the fault is due to a failed link or a failed node is made during network reconfiguration.

3.1.2.3 Process Name:	Network Reconfiguration
Inputs:	I/O Network Identifier Network Topology Root Links Active Root Link Network Configuration Network Status Error Analysis Report
Outputs:	Network Status Network Configuration Root Link History
Requirements Reference:	I/O Network Functional Requirements, Section 2.4.3
Notes:	None.

Description:

The purpose of this process is to reconfigure the network so as to restore error free communication to all reachable, non-failed nodes in the network. The action taken by this process will depend upon the type of failure reported in the Error Analysis Report. The fault identified in this report is actually a hypothesis about what is causing the errors on the network. This process in effect tests this hypothesis and then verifies that the network is again fully operational. Thus the network may go through several intermediate configurations before the reconfiguration process is complete.

There are six classes of faults identified by the Network Fault Analysis process described in Section 3.1.2.2. They are a root link failure, a babbler, a link or node failure, a node which transmits from a disabled port, a single node failure, and an undiagnosable failure. The Error Analysis Report indicates which one of these failure modes is presently causing disruptions on the network. Depending on the type of fault, it may also contain some additional information about the the source of the problem. A separate strategy exists to deal with each of these fault classes.

The reconfiguration process is considered complete when the node status chain is executed on the reconfigured network and does not detect any errors. The backup strategem for dealing with error phenomena which occur during a reconfiguration attempt but which are not anticipated is network regrowth.

In general, reconfiguration strategies are designed to deal with both active and passive faults in the hardware which makes up the root link. In particular, this includes the IOS, the active ports of the root node, and the cable between the IOS and the root node. Passive faults are characterized by the non-retransmission of data, sort of a barrier or obstacle to data flow in the network. A disconnected cable is an example of such a fault; data cannot be retransmitted over this cable but transmission between other connections in the network is not affected. Active faults are characterized by the disruption of data flow in the network beyond the boundaries of the failed component itself. An IOS with a transmitter stuck on high is an example of this type of fault; the stuck on condition is retransmitted throughout the network, masking or disrupting transmissions between network connections remote from the failed part. Since the same error conditions generated by a broken root link could also be generated by an IOS stuck on high, the reconfiguration algorithm must identify the specific cause of the problem so as to effect a repair. In the case of a passive fault, this means establishing another root link. In the case of an active fault, the faulty component must also be isolated.

A subprogram called Switch Root Link is designed to reconfigure a network in the presence of a root link failure. Prior to establishing a new root link to the network, the Switch Root Link subprogram updates the interface status of the channel connected to the failed root link to reflect the cause of the failure, a failed IOS or a failed channel. It also increments the error count against the failed root link in the root link history. Next a survey of the spare root links which are available to this network is conducted. An interface whose status is failed because of a faulty IOS is not a considered to be spare. However, the status of an interface which is failed due to an FTP channel failure may be upgraded to available if GPC FDIR now reports the channel as back online. The survey provides a prioritized list of spare root links, the lower the error count in root link history the higher the priority of a given spare.

If no spare root links are found but at least one root link is connected to a failed FTP channel, the Network Manager process suspends itself waiting for the outcome of GPC FDIR attempts to bring the failed channel back online. As a last resort, if no channels are brought back after a reasonable delay, the Network Manager tries to regrow the network. This will cause even failed root links to be tried one more time.

However, in the case where spare root links are available, each is tried in turn until a non-failed root link is found or until another type of fault is diagnosed. First, the new interface is used to execute the node status collection chain. Next, the node configuration table is updated to reflect the new root link; some outboard ports will become inboard and vice versa. Using this new configuration, an error analysis of the results of the node status chain is performed. If the analysis is unsuccessful, indicating the presence of an undiagnosable failure mode, the network is regrown. If no errors are detected, as would be expected in the case of a passive failure involving only the IOS, the inboard port of the root node, and the cable between them, the reconfiguration process is complete and the status of the new root link is marked active. If a root link failure occurs, the status of this root link is marked failed and the next available root link is tried. Finally, if either a single node failure, a babbler, or a link or node failure is detected, the root link switch is considered successful but the reconfiguration process is not yet complete. In this case, the reconfiguration process starts over again from the beginning, with a new root link but this time dealing with a new fault. This would be the case for an active fault such as a babbling IOS or a passively failed root node which now must be removed from the network so that service to nodes downline of it can be restored.

When a babbler is detected in the network, the network is regrown using the fast grow option since the detection and isolation of a babbler does not require any diagnostic testing. However, if the fault analysis has not been able to diagnose the failure mode, the network is regrown without the fast grow option since the diagnostic tests uncover failure modes which may produce unanalyzable error patterns, such as nodes which respond to the addresses of other nodes and nodes which respond late.

A subprogram called Repair Link or Node Failure is called to handle network reconfiguration when the Error Analysis Report indicates the presence of a failed link or node. Since a failed node generates the same error pattern as a failed link, this subprogram must determine which fault has actually occurred and reconfigure the network accordingly. The Error Analysis Report contains the node number of the node suspected to be failed, its inboard port and a list of nodes which are unreachable as a result of this failure. It is first assumed that a link has failed. The failed link is disconnected and an attempt to reach the failed node, i.e. the node immediately downline from the link, is made by using any spare ports on that node which are adjacent to nodes not in the failed node list. The chain used to reconnect this node to the rest of the network contains three transactions instead of the usual two. The first two transactions enable the ports on either side of the new link; the third transaction disables the former inboard port of this node in case the node adjacent to that inboard port is a babbler. When this strategy fails to restore communication with the failed node (possibly because no spare ports are available), data is assembled which will allow each branch of the failed tree to be reconnected to the active network. This data consists of a list of nodes for each branch of the failed node, i.e. a separate list for each set of nodes which lie downline of each of its outboard ports. Only one successful connection to any spare port on a branch needs to be made in order to restore communication to the entire branch (and possibly to the failed node and all other nodes in the failed tree). Again a three transaction chain is used, this time for a different purpose. The first two transactions enable the ports on either side of the new link while the third transaction attempts to obtain status from the failed node. If the failed node correctly returns its status, the repair is complete and the absence of errors is verified by collecting status from every node in the network. If the failed node is still not reachable, the port connecting this node to the present branch is disconnected and the proper functioning of the newly enabled link is verified. Then all the nodes on this branch are removed from the failed node set. The net effect of this process is to restore communication with all reachable nodes in the network while isolating the failed node. As communication to each branch is restored, the possible pool of spare links increases. Thus if any branch was not connected because of a lack of spare links, this branch is retried whenever a connection to another branch is successful. Any nodes which are still unreachable at the end of this exhaustive process are assigned a status of failed.

If a node retransmits valid data from a port which should be disabled, the node must be removed from the network. This failure mode is distinguished from a babbler which is always transmitting a random bit stream or is stuck on one. When a babbling port is identified, the adjacent port of the neighboring node is disabled. This neighboring node will not retransmit from its other enabled ports anything received by the disabled port. Furthermore, the node will ignore any random bit patterns it receives. However, if the neighboring node receives a request for status addressed to itself on a disabled port, it will transmit its status from all its enabled ports, even though it does not retransmit the initial request. If this failed node is not removed, each time the manager asks for status from the node adjacent to this port, it would receive two valid commands to report its status. Only one response is expected. Once the first response is received, another node will be commanded to report its status. The second response of the node may interfere with the reply of a node whose transaction is later in the chain, making it appear that this next node has failed to respond correctly to a command. Once the failed node has been removed from the network, status is collected from the remaining nodes to verify that in fact the fault has been identified and isolated. If errors are still detected in the network, a full regrowth, with a complete set of diagnostic tests, is performed.

The subprogram which removes a node from the network is called Remove Failed Node and Reconnect to Trees. As the name implies, removal of a node is a simple matter if the node is a leaf; only the link connecting it to the network needs to be disconnected. This is accomplished with one chain. However, if the node is the root of a subtree in the network, the nodes downline from the failed node need to be reconnected to the network through alternate links.

If the node to be removed is the current root node, a new root link is selected from the spare root links in the network. Prior to beginning the reconfiguration of the network, the nodes downline of each outboard port of the failed node (i.e. the nodes on each branch of the tree emanating from the failed node) are added to a reconnection queue, one queue for each branch which will be isolated when this node is removed from the network. Each of these nodes is also added to a set of unreachable nodes. The link connecting the inboard port of the failed node to the rest of the network is then disabled. Next, a loop is entered in which an attempt is made to reestablish a connection to each isolated branch via a spare link to a node in the reconnection queue of that branch from a node which is still reachable, i.e. is not a member of the unreachable node set. Only one such connection needs to be made to restore communication to all the nodes in the branch. After the new connection is enabled, the link connecting the failed node to this branch is disconnected. As each branch is reconnected, the nodes in that branch are removed from the failed node set. If any branch is successfully reconnected, branches which were not connected during earlier attempts are tried again since more spare links become available as communication is restored to nodes in other branches. Thus this algorithm, while isolating the failed node, restores communication to every reachable node in the network. Nodes which cannot be reached because earlier failures have depleted the pool of spares are marked failed.

If a single node in the network has errors, the reconfiguration is handled by a subprogram called Reconnect, Remove or Regrow. This failure can occur if the failed node is a leaf node, if its retransmission function still works correctly but its status reporting capability is impaired, or if another node is responding to this node's address, making it appear that this node is failed. If the failed node is the current root node, before proceeding, a new root link is selected from the available spares, status is collected using this new root link, and a new error analysis is performed. The failed node is then isolated from the network, as described above in the discussion of Remove Node and Reconnect to Trees, however, care is taken not to address this node directly. When the node is isolated, this node is again queried for its status. If a valid response is received, indicating the presence of a node which responds to the addresses of other nodes, the network is regrown with a full set of diagnostic tests to isolate this faulty node. Otherwise, an attempt is made to find an alternate root to this node using any port except its previously failed inboard port. The configuration command sent to this node as part of the link enabling procedure will disable this failed inboard port.

If two attempts to reconfigure the network have not succeeded in eliminating errors and network regrowth has not yet been tried, then a full regrowth is called for. This is the back up reconfiguration strategy, used when all else fails.

Following the reconfiguration of a network, the Network Status is updated to reflect the current state of the network hardware. If any nodes have been isolated from the network as a result of the reconfiguration, the transaction for that node is removed from the status collection chain. A new set of commands to cycle a spare link are set up and the Network State is given the value Repaired.

3.1.2.4 Process Name:	Spare Link Cycling
Inputs:	I/O Network Identifier Network Topology Root Links Active Root Link Network Configuration Network Status
Outputs:	Network Status Network Configuration Spare Link Cycling Log
Requirements Reference:	I/O Network Functional Requirements, Section 2.2
Notes:	None.

Description:

This process determines whether or not spare links are operating properly by routinely using a spare link as an active link in the network. When a spare link is called up for service, an active link is retired. The spare links are said to be cycled through the network since each spare eventually serves some time as an active link. The algorithm does not insure that the ratio of the time spent in an active state to the time spent in an idle state will be the same for all links. It does however insure that every idle link spend some time in an active state on a regular basis. For a network with S spare links and a cycling period T, each link in the network will be active at least T seconds of every ST seconds of operation. Some links may never be taken out of service. Cycling spare links provides greater fault coverage than merely testing a link since all parts are exercised for longer periods of time. Spare links are cycled at a rate commensurate with the desired fault detection latency and the testing overhead.

To cycle a root link does not require any physical changes in the network. Only the identity of the active root link is changed. However, to cycle a link between two nodes does require a network reconfiguration. The node on one end is arbitrarily designated the spawning node and the node on the other end is designated the target node. If this results in the spawning node lying on a branch downline of the target node, their roles are reversed. First, the link connecting the target node to the node adjacent to its inboard port is disabled by commanding each node to disable the port corresponding to the link. (Since two nodes have at most one link in common, this adjacent node is not the spawning node.) Next, the configuration of the spawning node is modified so that the port adjacent to the target node is enabled while its other ports retain their original configuration. The target node is then reconfigured to enable the port adjacent to the spawning node. The four transactions which cycle the spare link are executed in one chain, because the link switch must take place as an atomic action on the network. Any one of these transactions alone would isolate some set of nodes from the rest of the network.

When faults are identified in the network, the Network Manager sets up a new set of transactions to cycle a spare link after every network reconfiguration. In the absence of faults, after a spare link is cycled, the commands for the next cycle are set up. In order to avoid a rotation between several links in the network which exclude some spare links from the cycling process, an object called the Spare Link Cycling Log is created with one entry for each link in the network. Whenever the network is grown or repaired, each entry is given a value of untested if it is idle or tested if it is active or failed. As the cycling process moves a spare link into active service, the corresponding log entry is marked tested. When all the links are tested, the log is reinitialized as described above and the process repeats. The spare links present at the start of a cycle are not in general the same set each time.

Although the Network Manager determines which link to cycle, sets up the necessary transactions and processes the status and data returned by those transactions, it does not execute the spare link chain. This is accomplished by the I/O Communication Manager which runs a set of spare link chains in parallel in all the networks of an I/O Service when time is available on that service. This does not require that the network be taken out of service since the Network Manager does not use the network directly. The input/output data for the spare link cycling chains is protected by a test and set locking protocol. When the chain is complete, the I/O Communication Manager signals the Network Manager who then analyzes the data and status produced by the chain. If errors are detected, the Network Manager can set a flag which indicates whether or not the error can be repaired by restoring the link that was last retired or whether the network needs to be regrown. When the I/O Communication Manager detects the error, it signals the Network Manager to repair the network. The Network Manager examines the value of this error flag before taking a repair action

3.1.2.5 Process Name:	Restoring Repaired Network Hardware
Inputs:	I/O Network Identifier Network Topology Root Links Active Root Link Network Configuration Network Status Restore Record
Outputs:	Network Status Network Configuration
Requirements Reference:	I/O Network Functional Requirements, Section 2.2
Notes:	None.

Description:

The purpose of this process is to upgrade the status of a network component from failed to either idle or active and, if necessary, to reconfigure the network to establish communication with the parts of the network which were unreachable because of faults in the component. The action taken by the Network Manager depends on whether the restored component is a node or a link. Restoring a link does not require a network reconfiguration; the status of the ports adjacent to the link are simply upgraded to idle. Eventually, the operation of the link will be tested by Spare Link Cycling (3.1.2.4). Restoring a node, however, does require network reconfiguration, since the node itself must be reconnected to the active network. Furthermore, the node must be configured so that network subscribers, DIUs and GPCs, adjacent to the node are reachable.

To restore a node, the Network Manager tries to establish a link to that node from some active, adjacent node. The node to be restored is the target node and the adjacent node is the spawning node. First, the status of all the ports on the target and of all ports adjacent to the target are marked idle. Attempts to reconnect the node which produce errors result in the status of the respective ports being marked failed. When a connection is established, the status of the respective ports are marked active and the status of the node is marked active. Finally, ports adjacent to DIUs and GPCs are enabled. If no errors are detected, the connection is left in place; otherwise, the connection is disabled. The Network Configuration and the Network Status are updated when the process is complete.

Network components are presently restored to service only upon the request of an operator. This request is channeled through the I/O Communication Manager. If the component is a node, the network is taken out of service until the repair is complete since the Network Manager uses the network to effect the repair. Once the repair is complete, the network is put back in service. The repair is complete when the failed node is back online or when all possible links to the node have been tried without success. If the component is a link, the network is not taken out of service.

3.1.3 Network Manager Utility Operations

The purpose of this process is to provide the Network Manager with easy access to frequently used operations. Two subprograms reconfigure the network, Enable Link and Disconnect Link. Another subprogram takes care of updating the status of components which are marked failed because they are adjacent to a failed node. A set of subprograms format messages to nodes according to the node requirements described in Appendix D, the Node Specification. A complementary subprogram converts the raw data in the node response to a form more readily usable by the Network Manager. A final subprograms generates a list of unreachable DIUs on a network for use by the I/O Communications Manager. These are discussed in the following subsections.

3.1.3.1 Process Name:	Enabling and Disabling Links
Inputs:	I/O Network Identifier Network Topology Active Root Link Network Configuration Target Node Spawning Node Target Port Spawning Port Contention Option Maximum Retries
Outputs:	Link Enabled
Requirements Reference:	I/O Network Functional Requirements, Section 2.4.1
Notes:	None.

Description:

The Enable Link subprogram tries to enable the link specified by the caller which connects the spawning port of the spawning node to the target port of the target node. It returns a boolean valued record which indicates whether or not the connection was established and if not, the type of error detected. The caller also may specify the maximum number of times to try to establish the connection if errors are detected and whether or not the chain to effect this reconfiguration is conducted with contention. The default is one try. This subprogram sets up configuration commands to the nodes by using the subprograms described in section 3.1.3.2. Only one port in each node has its configuration changed by these commands. The configuration of the other ports as specified in Network Configuration are left unchanged while the target port and the spawning port are enabled. The command to the spawning node is executed first. The subprogram called to execute this chain is discussed in Section 3.1.4.1. This subprogram returns a configuration report describing any errors detected during the execution of the chain.

The information in the error report is analyzed to determine whether or not the link is fully operational. A link which is enabled after an error is detected is exercised one more time before it is considered active as a coarse filter for intermittent failures. If errors are detected, retries are attempted up to the specified limit. If the link is operational, the link enabled flag is set to true; otherwise it is set to false, and the cause of the last error is returned in the field of the record reserved for that purpose.

Disconnecting a link is a simpler process. The configurations of the target port on the target node and the spawning port on the spawning node are set to idleport. The configuration of the other ports is left unchanged. Commands are set up as before but the order of the transactions is reversed, the command to the target node preceeds the command to the spawning node. Errors detected during the execution of this chain are not processed, however, the spawning node is queried for status (with contention) after the target node is disconnected. Errors on this test chain cause the attempt to disconnect this link to be repeated. A flag is returned to the caller indicating whether or not the link is successfully disabled.

3.1.3.2 Process Name: Formatting Node Messages

Inputs:	Configuration Lifetime Node Address Desired Port Configuration
Outputs:	Formatted Node Message
Requirements Reference:	Node Specification, Appendix D
Notes:	None.

Description:

This process converts the logical commands which the Network Manager sends to nodes into formatted messages which the nodes can interpret. The format required by the node is described in detail in Appendix D, the Node Specification. Since the underlying M680X0 based machine can operate most rapidly on data packaged as an integral multiple of an eight bit byte, the Network Manager uses objects represented in this way to specify information in a node message in order to improve its perfomance. However, the message which the node can interpret is densely packed; each bit represents an aspect of the command. This design reduces the amount of data transmitted serially over the network, further improving performance. For example the lifetime of a port enable command (for all future transmissions or for one response only) is represented by a byte of data for the Network Manager but by a bit within a byte of the node message. The additional memory used by the Network Manager to support this approach is negligible.

The Network Manager sends two types of messages to nodes: a reconfiguration command and a request for node status. Hence, this process provides two subprograms to generate these messages. The inputs to the first subprogram are the node address, the configuration lifetime, and the desired port configuration. The second subprogram only requires the address of the node which will receive the status command. The format of both messages is the same. The subprograms copy the node address directly to the respective field of the formatted message. They encode the address as required by the Node Specification. The command they generate always requires the node to reply with a valid response from its status buffer, to clear the status registers after this response, and to transmit the response with three residual bits. They also append the correct checksum value for the message as required by the communication protocol for the system.

3.1.3.3 Process Name: Recording Status Changes for Failed Nodes

Inputs:	Network Topology Failed Node
Outputs:	Network Status
Requirements Reference:	I/O Network Functional Requirements, Section 2.4.1, 2.4.3
Notes:	None.

Description:

This subprogram uses the Network Topology to identify the network components adjacent to each port of the failed node and to update the status of those components accordingly. If the adjacent element is a node, the status of the port adjacent to this node is marked failed. If the adjacent element is a DIU, the status of the DIU is marked unreachable. If the adjacent element is an FTP channel of the GPC hosting the Network Manager, the status of the corresponding interface is marked failed IOS. Finally the status of each port in the failed node and the status of the node itself is marked failed.

3.2. I/O Sequencer Utilities

The IOS is a complex piece of hardware. The correct operation of the IOS is made possible by making an accurate software image of its many parts and by providing software functions and procedures to simplify the use of its many capabilities. The IOS must be initialized before it becomes operational. Furthermore, during the initialization process, it is tested to ensure that is operating correctly. Finally, a set of subprograms is provided to support the activities of the Network Manager. These subprograms execute chains and analyze the error information provided by the IOS before passing data from the nodes to the Network Manager. IOS initialization, testing and support for the Network Manager are the functions of the software modules in this section.

3.2.1 Principles Of IOS Operation

In order to control the use of an IOS, a software bit for bit image of all its registers must exist. Software templates will also be available to generate programs which the IOS will use to execute chains. The primary function of the IOS is offloading the IOP from the low level aspects of serial communications. It is therefore undesirable to reload the dual ported memory with IOS programs and static data, i.e. data which does not change for each chain execution since this takes IOP processing time. Hence, the dual ported memory is organized to hold all necessary programs and data used by the IOS. While a good deal of information about the organization of dual ported memory is needed by the IOP software to control the operation of the IOS, it is not desirable to fill memory with an exact image of each IOS. This is not only wasteful but unnecessary since the various IOSs operate identically, even though they may execute different programs. A quick and easy way of accessing each IOS is to use a table of IOS/DPM pointers and a data type which reflects the organization of the IOS/DPM memory space. Such a table eliminates the need to generate these pointers in real time. Given that some irregularity in the way address bits may be used, such a table is the only timely way of obtaining this information. Finally, some low level IOS operations will be implemented as subprograms which are useful not only for initialization but later during real time operation.

3.2.1.1 Process Name:	IOS Data Types
Inputs:	Not Applicable
. **Outputs:**	Not Applicable
Requirements Reference:	I/O Network Functional Requirements, Section 2.3.1; IOS Specifications, Appendix C
Notes:	The main purpose of this "process" is to provide type templates for the easy manipulation of IOS registers and dual ported memory.

Description:

This process is responsible for defining the software representation of IOS hardware registers and type definitions for input/output data for node transactions which are stored in the IOS DPM. The IOS specification lists seventeen registers which are accessible by the IOP directly or which can be accessed by means of an IOS program. Of these, three are not currently needed to correctly and fully use the IOS for network communications. These are three registers belonging to the High Level Data Link Communications (HDLC) device: the HDLC Address Register, the HDLC Receive Holding Register, and the HDLC Transmit Holding Register. The Address Register contains the address that the device would use for comparison if on-chip address recognition were being used. On-chip address comparison is not being used, hence this register is not mapped into software. Similarly, the HDLC Transmit and Receive Holding Registers are not used directly since the IOS has programmed instructions for transmission and reception of data. Two registers, the Solicited Chain Pointer and the Unsolicited Chain Pointer, are sixteen bit registers whose type is defined in IOS Access Types. The remaining eight bit register types are defined and bit for bit specified here. They are the Chain Status Register, the Interface Command Register, the Interface Status Register, the Timer Limit Register, the Poll Address Register, the Poll-2 Register, the Time Register, the three HDLC Control Registers, the HDLC Interrupt Register, and the HDLC Status Register. Memorandum AIPS-86-32R in Appendix C gives the full specification of these registers. These register types will be represented as bit mapped records. A field of the record which is only one bit wide will be represented by a Boolean type. A field requiring more than one bit will be represented by a specific type corresponding to that field.

This process will also generate byte wide templates for register values which will be needed at run time. Generating specific bit patterns on most microprocessors is a labor intensive operation, requiring many machine instructions. Having frequently used templates available will improve performance and readability of the code. The following list is not exhaustive but indicates some of the templates that are needed.

template	register
Disable_Timer	Timer Limit Register
IOS_Poll_Addr (derived from the GPC Address)	Poll Address Register
Disable_HDLC	HDLC Control Register #1
Set_Auto_Flag_Mode	HDLC Control Register #2
Clear_Auto_Flag_Mode	HDLC Control Register #2
Prime_IOS	Interface Command Register
Stop_IOS	Interface Command Register
Execute_With_Poll	Interface Command Register
Execute_Without_Poll	Interface Command Register
Clear_Chain_Status	Chain_Status_Register

Finally, this process will specify record types for input and output data used by the network manager and test processes. These include a Chain Status Record for garnering performance data about a given chain. This is data that is stored away by the IOS during chain execution which can be used for future analysis. Also specified here are the Node Input Record, with fields for the five bytes of data the IOS stores for every receive input instruction it executes as well as the data from the node itself, and the Node Output Record, with a byte count field, a data field, and a field for transmission status as recorded by the HDLC device. The record for test data will be a compound record based on node input and output records as needed for the IOS test process. Finally, a record describing unsolicited input will be defined here. Unsolicited input on an I/O Network is an error condition. This type is provided to support detection of this error by other processes.

3.2.1.2 Process Name: IOS Instructions and Programs

Inputs: Not Applicable

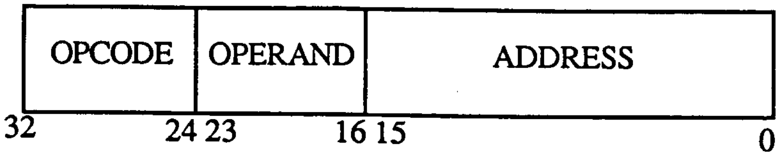
Outputs: Not Applicable

Requirements Reference: I/O Network Functional Requirements, Section 2.3.1;
IOS Specification, Appendix C

Notes:
Description:

This process is responsible for defining the software representation of IOS instructions and for specifying types which can be used as IOS program templates. In order to conduct communications on an I/O network, the IOS follows a program which has been previously stored for that purpose in the IOS/DPM.

The program consists of an ordered set of instructions. The format of the instructions is very specific and is described in detail in Appendix C. However, loading these instructions into memory is under software control. For many applications, as long as a compiler uses type templates in a consistent manner, the internal organization of RAM is not of general interest. However, the correct bit for bit layout of IOS programs is crucial to the correct operation of the IOS. Hence the internal organization IOS/DPM memory is different from other RAM memories. The template used to control the internal organization of this memory must be carefully spelled out for the compiler. For example, all IOS instructions must begin on an even numbered address, i.e. a word boundary.



opcodes: nop
output
unsolicited_input
solicited_input
move_immediate
move
branch

operand: 8 bit address of
IOS Register

Figure 16. IOS Instruction Format

There are seven basic IOS instructions: NOP, BRANCH, SOLICITED INPUT, UNSOLICITED INPUT, OUTPUT, MOVE, and MOVE IMMEDIATE. The format of these instructions is shown in Figure 16. Of these, only the NOP instruction requires no operands. The next four instructions require a single word (16 bit) operand. The three instructions dealing with input/output operations require the address of the data used by the instruction . The BRANCH instruction requires the address of the next instruction. The move operations have the most complex instruction formats. In addition to the sixteen bit address of the location to which the data will be moved, they also require the byte long (eight bit) address of the location whose contents are being moved or the absolute (immediate) value which is to be moved. In the case of the MOVE instruction, only byte long addresses can be moved. While this includes all readable registers, it does limit the general versatility of this instruction. Hence, only register contents are moved this way.

To simplify use of these instructions and to generate readable code, a set of mnemonically named constant instructions are defined in this process. Other processes can then use these constants as necessary. This method is also more efficient in that no CPU time be given over to assembling these templates at run time. However, this is not a major feature now since all IOS programs are loaded into the DPM during initialization. If later modifications require run time loading of IOS programs, this feature will become a time saving payoff.

Two types of constant instructions are defined, those using MOVE and those using MOVE IMMEDIATE. The MOVE instructions are defined as short instructions and specify a particular register to be moved. It is assumed that the destination address will vary and therefore will be added at initialization time. Thus these short instructions are only part of an instruction, since they require an additional operand for completion. The MOVE IMMEDIATE instructions are complete instructions which give a constant to be moved and a destination address. These are used to define frequently used operations. The following list of these definitions is not comprehensive, but identifies some of these instructions.

short instructions defined as constants

read_csr

read_isr

read_sys_time

read_ir

read_sr

full instructions defined as constants

stop_timer

enable_rcvr_only

enable_rcvr_and_xmitter

set_3_res_bits

set_auto_flag

start_chain_with_poll

While individual instructions are important, these instructions must be grouped together to create useful IOS programs. To this end some program templates are defined in this process. Other processes can use these program stubs to build larger IOS programs.

The program stubs which are defined here are a chain header, a node transaction, an array of node transactions, an end of chain program, IOS idle program, and a node chain program. A complete IOS program consists of a header, a chain of transactions, and an end of chain routine.

The header is used to store some initial data about the chain such as the time and the initial value of the HDLC status register and to initialize some IOS registers such as the poll priority registers and the HDLC receiver. It also contains the instruction which causes the IOS to poll for use of the network. The chain header ends by branching to the first transaction stub.

Each node transaction stub controls the execution of one I/O exchange with a node. It waits the required bus quiet time (currently 256 microseconds), enables the HDLC auto flag mode, sends an output frame, enables the timer to detect no response conditions(currently the timeout value is 512 microseconds) and receives the node response. Each transaction stub ends with a branch to the next stub. It is this resemblance to a linked list that spawned the use of the name "chain" for the IOS program.

In order to stop the IOS chain execution, an end of chain program is necessary. This stops the timer, disables the HDLC transmit and receive functions, stores some statistics, and finally sends the IOS to its idle mode of operation where it executes the IOS idle program. The last transaction in the chain branches to this end of chain program.

The IOS idle program is simply a way to have the IOS execute a program using the unsolicited chain pointer as its program counter (PC) while it waits for a command to execute a solicited chain. The later uses the solicited chain pointer as its program counter. This takes advantage of the ability of the IOS to transition smoothly from one PC to the other. The idle program is simply an infinite loop which executes an untimed unsolicited input instruction, followed by a branch to this instruction. When a solicited chain is ready, the transition from one PC to the other is effected by writing to the Interface Command Register. The end of chain program terminates by writing a command to the Interface Command Register to change PCs.

Although the DPM consists of random access memory, the program portion is relatively static. Only the data portion, and then only dynamic data, will be rewritten either by the IOP (for output packets) or the IOS (for input packets). Thus these programs are loaded into the DPM at initialization time and do not need major modifications at run time. The modifications which do need to be made will be discussed further in section 3.2.1.4.

3.2.1.3 Process Name:	IOS Access Types
Inputs:	I/O Network Identifier IOS Identifier Channel Selection DPM Partition Relative DPM Address System Address Long DPM Address
Outputs:	Long DPM Address System Address Relative DPM Address
Requirements **Reference:**	I/O Network Functional Requirements, Section 2.3.1; IOS Specification,Appendix C

Notes: The action of this process is necessitated by the way in which address lines control access to devices in the LMN region of memory. The IOS is a device in that region.

Description:

This process is responsible for mapping logical entities such as IOS identifiers and IOS registers into physical address bits. IOS/DPM accesses are made by means of pointers. The addressing scheme is fairly complicated. This process makes addresses for accessing specific IOS/DPM pairs readily available so that this type of bit assignment does not need to be done in real time.

A good deal of information about the behavior of read/write operations on an IOS/DPM is encoded in the address used during the read/write access of this device. Figure 17 is intended to help with this discussion. Each DPM currently has 8K bytes of storage. Within a DPM, this memory is divided as follows. The first 32 locations are specific IOS hardware registers. In some cases there is further address decoding based on the value of the read/write control line. The remaining 8,160 locations are dual ported RAM. One unusual aspect of the IOS address space is that the actual address used to access a given location in dual ported memory depends on the side originating the access. From the IOS side of the DPM, the locations are accessed by thirteen bit addresses ranging from 0 to $1FFF_{16}$. The FTP is currently based on a Motorola M68010 microprocessor and the address lines referred to in the following discussion are the M68010 address lines. Figure 17 will help with the following discussion. As one would expect, address lines A0 to A12 are decoded to designate one of these byte wide locations. However, when accessed from the FTP side, the highest order address bit is not at A12 but rather at A15. Thus, from this side, the DPM is effectively partitioned into two discontiguous regions. The lower 4K bytes, including the IOS registers, are accessible when A15 has a value of zero; the upper 4K bytes are accessible when A15 has a value of one; decoding A0 to A11 selects one byte within a given 4K region. To make use of this feature, the DPM has been divided into two functional regions as well. The lower 4K are used to hold programs and data for the I/O Network Manager while the upper 4K are used by the I/O Communications Manager for user programs and data. Care must be taken when converting FTP addresses for use by the IOS that the value of A15 is transferred to A12. This conversion is handled by a function which takes a Long DPM Address and returns a Relative DPM Address for use in the IOS programs which are stored in the DPM. This function copies the lower twelve bits of the Long DPM Address to the lower twelve bits of a sixteen bit word. It then copies A15 from the Long DPM Address to bit 13 of the word. Finally, it zeroes the upper three bits of the word thus forming an address which occupies two bytes and can be used as an operand in IOS instructions which require an address.

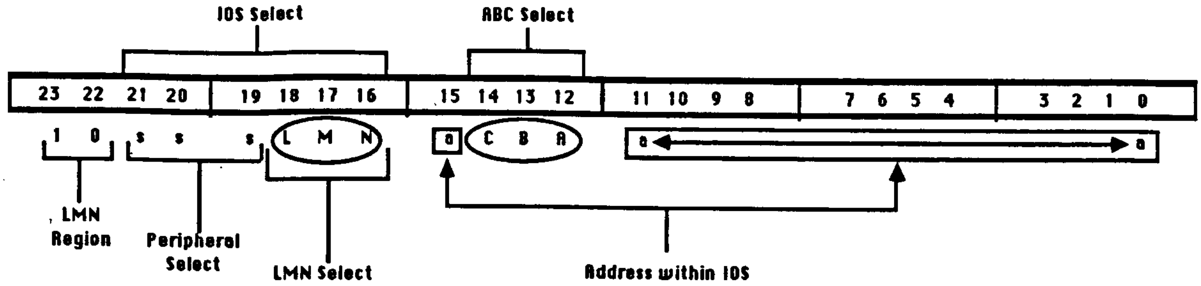


Figure 17. FTP Address Lines

From the FTP side, the remaining eleven address lines, A23 to A16 and A14 to A12, must be decoded to select a particular IOS or set of IOSs. The function of address lines A23 and A22 are very straight forward. When A23 has a value of one, the shared bus is selected. When A22 has a value of zero the LMN region of the shared bus is selected. Since the IOS is part of the LMN region of the shared bus, pointers to an IOS/DPM must conform to this bit usage.

Within the LMN region, the address space of each FTP channel can support 24 IOS, as shown in Figure 18. Selecting one of these 24 IOSs is controlled by address lines A21 to A16. A21 to A19 are the peripheral select bits and they can take on any value from zero to seven. A18 to A16 are the LMN bits and for an IOS exactly one of these bits must have a value of one; the other two must be zero. Finally A14 to A12 are the ABC channel selection bits. While the value of A23 to A16 select a particular IOS within each FTP channel, the FTP channel or channels which will actually transfer data to its attached IOS is determined by the ABC bits. To select a channel, the bit corresponding to that channel is set. At least one of these bits must be set, but more than one can also be set. When one bit is set, only that channel's selected IOS is accessed. When two or more ABC bits are set, the selected IOS in each designated channel is accessed simultaneously within its respective channel. While the address space itself will support twenty-four IOSs, the backplane is presently capable of containing only two IOS boards. These will both reside in the L region.

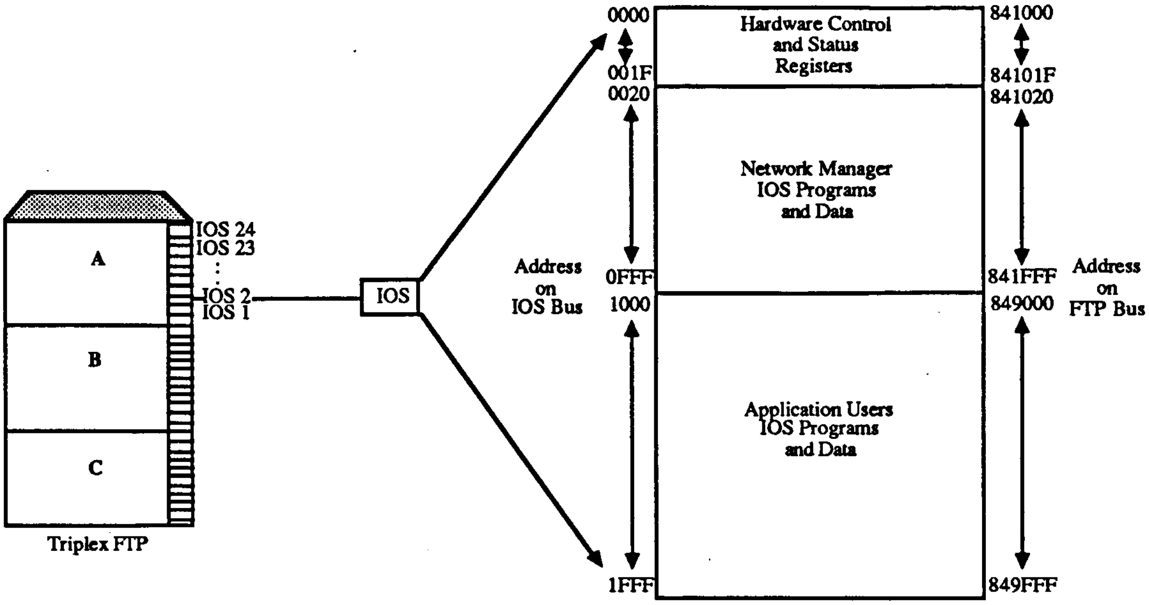


Figure 18. IOS/DPM Functional Memory Map

The ability to access corresponding IOSs in different FTP channels simultaneously can afford some performance gains when working with networks with multiple root links. For reliability considerations, these root links are always connected to different channels. Furthermore, within their respective channels these root links are assigned to corresponding IOSs (see Appendix B, Network Operating Rules). Although each IOS operates independently, all the IOSs to a given network contain exactly the same set of programs and outgoing data. Although only one IOS to a network is active at any given time, the inactive or spare IOSs must be ready to take over network operation in the case of a failure of the active IOS. But only the active IOS holds valid incoming data after executing a chain of network transactions. Therefore, when writing data or programs for a given network, all its IOSs are written to simultaneously. When reading data from a given network, only the active IOS is read.

To support the features described above, read and write operations for the IOS will use distinct pointers with the proper functionality encoded in the ABC selection field. If the bit is set, data is transferred. Therefore, pointers used for writing data to an IOS/DPM will have the bits set which correspond to all its root links. Only one write pointer value needs to be generated. Pointers used to read data from an IOS/DPM have exactly one of the ABC bits set. Thus, a read pointer for each root link in a network needs to be calculated.

IOSs are simplex devices which do not contain source congruent data. The issue of source congruency of inputs and voting of outputs in order to mask errors needs some discussion. When reading a given IOS, it is necessary to distribute the value read to all the FTP channels by means of a data exchange in order to maintain source congruent data in all

FTP channels. The channel whose IOS will be read has the corresponding ABC bit set. The actual distribution of the data to the various FTP channels is controlled by the LMN data exchange default registers, which have previously been set up to perform a self to all exchange. This is an implicit data exchange which does not require explicit use of the transmit or receive data exchange registers. When writing data to the IOS, the value written must be a voted value to avoid transferring bad data from one FTP channel to the network. Voted values mask any errors from one channel. Since there is not an analogous implicit data exchange mechanism available for write operations, writing a voted copy of data to the IOS requires explicit use of the data exchange registers. This will be discussed in Section 3.2.1.5.

To represent all this functionality in software, a data type called Long DPM Address is specified by this process. This type provides a bit for bit template of the IOS address fields. Values generated from this template are then stored in a table which has separate read and write entries for all the root links to the various networks connected to a given GPC. This table will be described in more detail below.

The type Long DPM Address is represented as a record with each set of address lines mapped to a specific field of the record. The fields of the record are: Bus, IOS, DPM_Half, Channel, DPM_Address. The Bus field(two bits) is assigned a constant value of 10_2, which designates the shared bus as described above. The six bit IOS field is used to select one of twenty-four IOSs. To ensure that the IOS select field has a valid value, an enumerated type is defined for this field with twenty-four possible literals. Each enumeration literal within the type is then assigned a specific two digit octal value which will decode as one IOS selection. To facilitate their use, these literals in turn are mapped to the twenty-four valued logical IOS identifier type. For the present system, only two of these twenty-four literals will actually map to physical IOS boards: 04_8 and 14_8. The low order octal digit, 4, indicates that both IOSs are in the L region of the shared bus. The upper or lower 4K of the DPM is represented as a bit wide Boolean valued field. Each ABC bit is an element in a packed Boolean valued array. The array is indexed by the type representing the logical identifier of the channel. The DPM_Address field is the eleven bit address of one byte within a 4K DPM region.

During initialization, a table of DPM addresses is generated. Each network to which a GPC is connected has an entry in this table. Each entry consists of two write address values, one for each half of the DPM and two times R read address values, where R is the number of root links to the network from the GPC. The write addresses are designed to take advantage of the fact that root links to a given network must always come from corresponding IOS selections in each channel and that there is at most one root link to a network from a given channel as described above. Thus, it is possible to write programs and data to all IOSs (i.e. root links) of a given network simultaneously. The read addresses allow congruent copies of data from the active IOS in a network to be distributed to all FTP channels implicitly, without the overhead of an explicit data exchange.

A sample entry in this table is shown below. This network has two root links, one from channel A and one from channel B. It uses the IOS slot with peripheral select 0 from layer L. Thus, its IOS select field is 04_8. Figure 19 shows a read access memory map of the IOS in channel A.

POINTER FUNCTION	CHANNEL	LOWER DPM ADDRESS	UPPER DPM ADDRESS
Read Access	A	841000	849000
Read Access	B	842000	84A000
Write Access	All	843000	84B000

Figure 19. IOS Read Access Memory Map

3.2.1.4 Process Name:	IOS Dual Ported Memory Usage
Inputs:	DPM Pointer Long DPM Address I/O Network Identifier Channel Number
Outputs:	Initialized IOS/DPM for Network Manager
Requirements Reference:	I/O Network Functional Requirements, Section 2.3.1; IOS Specification, Appendix C
Notes:	None

Description:

This process is responsible for initializing the IOS/DPM for the network manager. A logical organization of the IOS/DPM is superimposed on the physical memory space by defining a record type, DPM_Record. A variable which is an access type to this record can be assigned the base address of a specific dual ported memory. This base address can be obtained from the table described in section 3.2.1.3.

The DPM_Record will have a specific field for each IOS control and status registers. Other fields are defined to contain the following programs: a test program for the IOS, a status collection program, a node reconfiguration program, an end of chain program, and an idling program. Finally, the following fields are defined to hold data : chain status data, data for the IOS test program, input/output data for the node status collection and node reconfiguration programs, and a data field to trap unsolicited input. Using this record will make logical, rather than absolute, references to fields of the IOS/DPM possible. Furthermore, if the language supports the ability to obtain an address of a given field of a record, this address can be processed to generate the corresponding address needed by the IOS. For example, the thirty-two bit address used by the FTP to access a node output packet can be converted to the sixteen bit address used as an operand by the IOS send instruction, thus automating and simplifying the software needed to generate IOS programs.

This process will provide subprograms to initialize the IOS/DPMs which are connected to the host GPC. There are four steps in the initialization of an IOS for the I/O Network Manager. The four steps are: IOS related hardware initialization, writing the IOS programs to the DPM, writing static output packets for node status collection to the DPM, and writing static output packets and clearing memory to hold input packets for the IOS test program. Each of these will be considered in turn.

Hardware Initialization

The IOS hardware is initialized by writing to specific control registers. Values are written to accomplish the following: stop and prime the IOS by writing to the Interface Command Register(ICR), zero all non-register DPM memory, zero the Solicited Chain Pointer(SCR), write the starting address of the idling program to the Unsolicited Chain Pointer (UCR), disable the timer by writing to the Timer Limit Register(TLR), set the value of Poll Register 1 to the IOS_Poll_Address, disable the autoflag mode of the HDLC by writing to the HDLC Control Register #2 (CR2), disable both transmission and reception by the HDLC by writing to the HDLC Control Register 1 (CR1). The initialization of the hardware registers is performed simultaneously on all IOSs connecting this GPC to the network. The subprogram performing this initialization is passed the pointer of the IOS or set of IOS to be initialized. If there is a channel failure and recovery later in the lifetime of the system, the pointer value passed to this routine will allow it to reinitialize only one channel.

IOS Programs

The DPM holds four programs for use by the I/O Network Manager and one test program executed at power on. These programs are the end of chain program, the idling program, the node chain programs, and the IOS test program. The end of chain program stops the timer, disables transmission and reception by the HDLC, saves the value of the time byte and the Chain Status Register (CSR) in the Chain Status data area, and writes a command to the ICR indicating that an unsolicited chain should be executed. This causes the IOS to begin executing instructions from the UCP which is pointing at the idling program. It also causes the Chain Complete bit in the CSR to be set.

The idling program starts an unsolicited input instruction, pointing at the buffer used to trap unsolicited input and then branches to itself. During this program the IOS is waiting for unsolicited input which should of course never come. It really is idling, waiting for the command to execute a solicited chain.

Each node chain program starts with a header and is followed by a linked list of transactions. The header has instructions which save the value of the time, the HDLC Interrupt Register (IR), and the HDLC Status Register (SR) in the Chain Status data area. The header then has instructions to write to CR3 commanding the HDLC to use (and expect) three residue bits per HDLC frame, to set the poll priority for this chain (manager chains are given the highest priority), to start a chain with a poll by writing to the ICR (This is the default value; the status collection chain always starts with a poll, but the reconfiguration chain may not.), and then branching to the first transaction in the chain.

Each node transaction executes the same series of instructions. First the HDLC is commanded to disable reception and transmission. Then the TLR is programmed to go off in 256 microseconds followed by a solicited input instruction with the address of the unsolicited input buffer. The effect of this combination of instructions is to produce a delay of 256 microseconds. The network nodes need this much "quiet" time on the serial bus between transactions for proper operation. Then the TLR is disabled. Next the HDLC is commanded via CR2 to enable auto flag and via CR1 to enable its transmitter. Next a send instruction is given with the address of the output packet belonging to this transaction as an operand. A no_op instruction is required by the IOS following each send instruction. Autoflag is then disabled (by writing to CR2) and the HDLC receiver is enabled while the transmitter is disabled (by writing to CR1). Next the value of IR is copied to the transmission status field of the output packet for this transaction for later processing. The TLR is set to go off in 512 microseconds followed by a receive instruction whose operand is the address of the input record associated with this transaction. The last instruction in a transaction program is a branch instruction to the next transaction in the chain or to the end of chain program.

The last program to be initialized is the IOS test program. The purpose of the test program is to verify the proper operation of the IOS and its attached root node. It consists of a header and two node transactions. It will be discussed in more detail in section 3.2.3.2.

The initialization of the IOS programs is performed simultaneously on all IOSs connecting this GPC to the network. The subprogram performing this initialization is passed the pointer of the IOS or set of IOSs to be initialized. If there is a channel failure and recovery later in the lifetime of the system, the pointer value passed to this routine will allow it to reinitialize only one channel.

Static Output Packets for Node Status Collection

Collecting status from network nodes is a periodic function. While there may be occasion to deselect some of these transactions due to the failure of a node, the chain of transactions used to collect node status is essentially static. To deselect a transaction from the chain, the operand of the branch instruction which preceded the transaction in the list is modified to point to the next transaction. This modification affects the program but not the data associated with the transaction. In the status collection chain, each node in the network is sent a command asking it to send back the contents of its status registers. Then a response is awaited and,in the absence of faults, received under the direction of the program. This command sent to the node never needs to be changed. Hence the output packets containing the node command can be written once at initialization time and never written again. In contrast to this chain, the node reconfiguration chain has data packets which change each time the chain is executed. Hence, theses output packets are not initialized except for the byte count field. This field is always a fixed number for all node output packets and writing it once at initialization does afford a small performance gain.

The initialization of the output packets for the node status collection chain is performed simultaneously on all IOSs connecting this GPC to the network. The subprogram performing this initialization is passed the pointer of the IOS or set of IOSs to be initialized. It also needs the network identifier of the network whose IOSs are being initialized so as to be able to obtain information about the addresses of the nodes in the network. If there is a channel failure and recovery later in the lifetime of the system, the pointer value passed to this routine will allow it to reinitialize only one channel.

Static Output Packets for the IOS Test Program.

The IOS test program also sends output packets, but only to the root node. Since this test is only performed once at power on or system restart, its output packets are also static and only need to be written once to the DPM. These output packets are directed to the root node connected to each IOS.The test input records, like all node input records, contain the node responses to the commands in the test output packets as well as status information about the transaction which the IOS appends. Since this test is run at startup, these records can be initialized here to a bit pattern which will be written over by the IOS when a valid node response comes in.

Unlike the other data and programs described thus far, the data in the test output packets is not identical in every IOS which connects this GPC with a given network. This is because the output packets in the test program go to the root node only, and the root node from each IOS has a unique address on the network. Thus, when writing all the programs and the data for the node status collection chain, the DPM pointer which is used is the one which allows simultaneous writing to all the IOS/DPMs connected to the network. The subprogram which initializes the IOS test data packets needs to be passed the network identifier and the channel number of the IOS which is being initialized.

3.2.1.5 Process Name: IOS Low Level Utilities

Inputs:

Byte Pointer
Byte Count
HDLC Status Register
Initial Poll Priority
Interface Command Register
Unvoted Byte

Outputs:

Valid Checksum Boolean
Residue Bit Count
IOS Instruction
Interface Status Register
Voted Byte

Requirements Reference:

I/O Network Functional Requirements, Section 2.3.1;
Node Specification, Appendix D

Notes: None.

Description:

This process is responsible for providing users with some low level utility routines which hide the complexity and detail associated with such routines. Five routines are provided. They perform checksum verification, residue bit calculation, specialized IOS instruction generation, register type conversion, and voting output data. Each will be discussed in turn below.

Each input packet received by the IOS from a node has a checksum appended by the node. A valid checksum has the following characteristic: when it is added to the sum of the HDLC address, control fields and data fields in the packet the final sum modulo 256 is zero. This serially transmitted data is copied by the IOS into contiguous locations in the DPM. When the data is copied to onboard FTP memory, the data mapping is left intact. A routine, Valid_Checksum, will return a boolean value of true if the input packet from the node passes the above test and a boolean value of false otherwise. Since the data over which the checksum is computed is located in contiguous memory, only the address of the input packet and the number of bytes in the packet need to be passed as parameters to this general purpose routine. Tc avoid use of the mod operation which may require a time consuming divide operation, each partial sum is compared to the modulus. If this partial sum is greater than or equal to the modulus, the modulus value is subtracted from the total.

The HDLC protocol used for communication between a node and an IOS requires the transmission of three additional bits after the data field of each frame. These are called residual bits. After the reception of a frame, the HDLC Status Register value can be examined for the number of residual bits transmitted. A subprogram is defined which processes the Status Register value and returns the number of residual bits transmitted with an input packet.

When an I/O Network is shared among several GPCs, it is called a regional network. These GPCs contend for use of the network. The winner of a contention is determined on the basis of a priority scheme. The priority which decides network access is based in part on the priority of the message it wishes to send. This is a three bit field of Poll Register 2. A subprogram is defined which converts the logical message priority into the correct byte wide value to be written to Poll Register 2. When an IOS cannot win a poll, it will increment this three bit field prior to engaging in another contention. This dynamic aspect of message priority is intended to prevent the possible exclusion from the network of a low priority message; it is completely under the control of the IOS and requires no software intervention except for the loading of the initial value as an operand in the IOS program.

The Interface Status Register(ISR) and the Interface Command Register (ICR) occupy the same address in memory. They are selected by means of the value of the read/write control line. The former is read only and the later is write only. The DPM_Record format does not allow this type of hardware representation. Since it is referenced more frequently, the field has been designated as the ICR within the record. From the point of view of the software, this is a read/write register. To interpret the value read correctly, the bit pattern within the byte must be evaluated with a proper template. A subprogram which makes this type conversion is provided by this process.

Within the physical memory of an FTP channel, there may exist a latent fault such that the value stored in a given location is not congruent with the value stored in the same location in the other FTP channels. If this channel were the host of the active root link of a network and the faulty location held the value to be written to the IOS/DPM for transmission on the network, the faulty value would be stored in the DPM and then be sent on the network with possible adverse effects for the entire system. To guard against this problem, each byte of data which is stored in the DPM is first subjected to a bit for bit majority vote among the FTP channels. This voted value is then written to the DPM. This provides a high degree of confidence in all output data. The checksum included in all output packets protects against possible errors in the DPM itself. The checksum is calculated synchronously by each channel of the FTP on the local copy of the packet, voted across channels, and then written to the DPM. While the Frame Check Sequence (FCS) which the HDLC appends to outgoing packets detects bit flips during the transmission of a message, the checksum guards against errors that exist in the DPM itself. When a message which does not have a valid checksum or a valid FCS is sent to a node, it is not processed by the node. Thus the voting of outputs masks faults originating in FTP memory, the FCS detects errors during transmission, and the checksum covers faults which may exist in the IOS/DPM. A function which takes a byte from memory, subjects it to a bit for bit majority vote by sending it through the data exchange mechanism, and writes the voted value to the DPM is provided by this process.

3.2.2 Process Name: IOS Initialization

Inputs: Channel Identifier

Outputs: Reinitialized IOS
Initialization of Every IOS

Requirements Reference: I/O Network Functional Requirements, Section 2.3.1

Notes: None.

Description:

This process is responsible for coordinating the initialization and test of every IOS which is connected to a GPC. This takes place during system startup. If GPC FDIR determines that an FTP channel is failed due to a transient fault, it will bring that channel back online. This process is responsible for reinitializing the IOSs attached to the FTP channel after such a reconfiguration.

The test and initialization sequence is performed on each network connected to the GPC. It proceeds as follows:

1) the IOS Simple Response Test is performed (3.2.3.1)
2) the Address Line Test is performed (3.2.3.1)
3) the IOS Registers are initialized (3.2.1.4)
4) the IOS/DPM is loaded with programs and data for the I/O Network Manager (3.2.1.4)
5) the Correct IOS Operation Test is performed (3.2.3.2)
6) the Interface Status is written to I/O Network Status (3.4.1)

The reinitialization of an IOS is accomplished by means of the following sequence of steps:

1) the IOS Registers are initialized (3.2.1.4)
2) the IOS/DPM is loaded with programs and data for the I/O Network Manager (3.2.1.4)
3) the I/O Network Manger programs and data are updated (3.2.4.3)

3.2.3 IOS Testing

This process is responsible for determining whether or not the IOS hardware which is connected to a GPC is functioning properly. The tests cover the operation of the IOS, the DPM memory, and the root node connected to the IOS. If any faults are detected in this hardware, the information is logged in the IOS Error Log and the status of the network to which this IOS belongs is updated by marking the corresponding interface status as Failed_IOS or Failed_Channel, whichever is appropriate.

3.2.3.1 Process Name: DPM Memory Tests

Inputs:

I/O Network Identifier
Channel Number
Starting Address of DPM Memory Block
Ending Address of DPM Memory Block
Network Interface Status
Channel Identifier

Outputs:

Network Interface Status
Results of DPM Word Test
Results of DPM Block Test
Results of DPM Memory Tests
Results of Channel OK Test

Requirements Reference:

I/O Network Functional Requirements, Section 2.3.1;
IOS Specification, Appendix C

Notes: None.

Description:

This process is responsible for testing the Dual Ported Memory (DPM) associated with each IOS connected to a GPC. Some of the tests are conducted at system startup and others are conducted during normal operations.

The three tests conducted at startup verify that the IOS/DPM board is plugged into the backplane, that the hardware address lines are connected correctly and are fully operational, and also that the individual memory devices can pass read/write pattern tests. Of those tests conducted during normal operations, two are used as a diagnostic tool to further isolate the cause of an error after the error has been detected by other software and one is used as a routine diagnostic procedure, checking for latent DPM memory faults. The two tests used to further isolate detected faults determine whether or not a particular FTP channel is still synchronized with the other channels and whether a block of DPM memory has failed such that it cannot pass a read/write pattern test.

Each of the tests conducted at startup is conducted by a specific subprogram which tests all the hardware associated with a particular network. However, all the subprograms have certain operational features in common. One common feature is the use made of FTP channel status information available from the GPC FDIR process and similar information obtained from the channel failure detection subprogram provided by this process itself. If an FTP channel is failed, the memory tests conducted on the IOS/DPMs connected to that channel will appear to fail. Thus, prior to conducting any memory tests, the FTP channel status is evaluated. If the channel is failed, the interface status of the IOS is marked Failed_Channel and this information is logged. Similarly, at the conclusion of any test, if errors were detected, FTP channel status is again evaluated. If the channel failed during the test, thereby resulting in a false diagnosis of the cause of the error, the interface status is updated to reflect the true cause of the error and this information is logged. Another common feature among the three startup memory tests is the way in which pointers are used to access the DPM. Whenever values need to be written to the DPM, a pointer value accessing all the DPMs connected to the network is used. However, whenever values need to be read from the DPM, a pointer value accessing only one DPM is used. These pointer values are obtained from the IOS/DPM Memory process. To expedite the testing process, the cumulative result of these tests is passed as a parameter to each subprogram. Thus, if a DPM has failed a previous test, the current test will not be conducted. For example, a DPM which has failed the address line test will have its status marked Failed_IOS and therefore the read/write pattern test for that DPM will not be conducted. Similarly, whenever a test detects a fault, the test underway is deemed complete; further testing for that particular fault condition is not necessary. The purpose of the tests is to closely identify a fault for maintenance purposes, but not to provide a comprehensive diagnosis of the nature of the fault. For example, the test will diagnose a failed DPM memory device, but it will not identify all the memory locations in the device which may have failed.

The first of the startup DPM memory tests is used to verify the presence of the IOS/DPM board in the backplane. Using a pointer which accesses all the IOSs in the network whose identifier is passed as an input parameter, this test writes a pattern to the solicited chain pointer of the IOS. The values are then read back one at a time using a read access pointer. If the pattern read does not match the pattern written, the IOS is deemed unreachable by the FTP. The error is logged, and the interface status of the IOS is marked Failed_IOS. If the pattern does match, no log entries are made and the interface status is not changed.

The second startup DPM memory test determines if the address lines to the IOS/DPM are wired correctly. The test is only performed if at least one of the IOSs belonging to the network whose identifier is passed as an input parameter still has an idle, i.e. non-failed, interface status. The interface status is passed as an input/output parameter. Using the write access pointer obtained from the IOS/DPM Memory process, a value based on the lower twelve bits of the address of each location is written to the DPM. This pattern is then read back on a channel by channel basis and verified. The test is then repeated using a pattern in which the upper and lower bytes of the previous pattern are inverted. This switch is necessary since certain address line errors would not be detected otherwise. Each of the lower twelve address lines decodes as one DPM location which can only hold eight bits of information. Thus, after the pattern has been written, some locations will contain the same bit pattern. The first pass uncovers errors in the lower eight address lines and the second pass in the upper four address lines. Errors in higher order address, i.e. lines twenty-four to thirteen, lines will be uncovered by the first pass. If the pattern read does not match the pattern written, the IOS is deemed to have an address line error. As in the previous test, the error is logged, and the interface status of the IOS is marked Failed_IOS. If the pattern does match, no log entries are made and the interface status is not changed.

The last DPM memory test is the read/write pattern test. In this test each IOS connected to the specified network is tested in turn. A pattern is written to one location and then read back. If the value read matches the value written, the next location is tested. If an error is detected, the interface status of this IOS marked Failed_IOS, the information is logged and the process is repeated with the next IOS/DPM in the network. For completeness, two patterns need to be used, one being the bitwise complement of the other. This will ensure that each bit in the memory can hold a value of one and as well as a value of zero.

The first of the two diagnostic tests used to further analyze other test results is the test which performs a read/write pattern test on a block of DPM memory. In this test the starting and ending addresses of the block are provided to the subprogram. No checks for channel failure are performed here. Also, this test is only conducted on one DPM at a time, the one selected by the starting and ending addresses. This test is used whenever other fault detection indicators give a positive result but the possibility of a channel failure is already ruled out. If no errors are detected in the designated block of memory, an indication that the test was passed is returned to the caller; otherwise an indication that the test was failed is returned.

The second of these tests is one which is used to detect an FTP channel failure. This test is accomplished by performing a From_X exchange on the channel under test, where X is A, B, or C. The From_X exchange causes a value to be written to the transmit register of the data exchange hardware. While all channels write to their respective transmit registers, only the copy from channel X is allowed to proceed to the receive registers in each channel. This exchanged value is then read back from the receive register and compared with the outgoing value. If the values do not agree, a channel failure is likely and an indication of this result is returned to the caller of this subprogram. Otherwise, an indication of no channel failure is returned. For this test to work, the data exchange hardware must be functioning properly and all channels must be operating synchronously. The purpose of this test is not to analyze this error since that analysis is the responsibility of GPC FDIR. Rather, this function provides a quick check as to the state of the FTP channel connected to an IOS. There are two main uses for this test. The first is to detect a channel failure before an IOS connected to that channel is used to execute chains which access I/O devices on a network. In the presence of such a failure, another IOS connected to the same network through a different FTP channel must be used since only the channel directly connected to the IOS can control the IOS. Detecting this condition early saves time, since such a failure will always result in the detection of errors in the I/O chain. These errors must then be processed and the chain repeated. All this unnecessary overhead can be avoided by the use of this test. The second use of this function is to help in the analysis of the cause of an error, once that error is detected by other means. Thus, if data read back from a transaction to a node indicates a checksum error, this fault could be due to the node itself, to the IOS/DPM memory location in which the checksum was stored, or to the failure of the FTP at the time the checksum value was read from the DPM to the FTP RAM. (Bit flips during transmission are screened for earlier in the logic chain and therefore are already ruled out as a cause.) Use of the two diagnostic tests described so far isolates the problem to one of these three causes.

The last diagnostic test is provided to allow background testing of the IOS/DPM when time for such a contingency function is available. Each word in the DPM memory is tested in turn, a new word with each call to the subprogram which executes the test. The network identifier and the channel number which uniquely identify the IOS/DPM to be tested are passed as parameters to this subprogram which conducts a read/write pattern test on the next location in the DPM memory. If the test fails, this result is logged and an indication of this result is returned to the caller. If the test is passed, an indication of this result is returned to the caller.

3.2.3.2 Process Name: Tests for Correct IOS Operation

Inputs: I/O Network Identifier
Network Interface Status

Outputs: Network Interface Status

Requirements Reference: I/O Network Functional Requirements, Section 2.3.1;
IOS Specification, Appendix C;
Node Specification, Appendix D

Notes: None.

Description:

This process is responsible for testing the proper functionality of each root link connected to a GPC. The root link consists of the IOS and the root node. These tests are conducted at system startup. The test is conducted by executing a program from the IOS which sends commands to the root node. The results of the program are then analyzed to determine if the results indicate a fully operational IOS and root node. The IOS functions which are tested include its polling capability, mode switching capability, timeout operation, ability to transmit and receive data, status and control register operation, and its overall capability to execute a chain, i.e. execute its instructions correctly. The node functions which are tested in the root node are its ability to disable its root port, to return its status only from an enabled port, to respond to reconfiguration commands and to reconfigure itself for one transaction only.

The test is performed on one IOS in a network at a time. The following sequence of events prepares the IOS for the actual test execution. The ICR is commanded to stop; this is an effective reset of the IOS. A chain is then executed without conducting a poll which sends one reconfiguration command to the root node instructing it to disable all its ports. This chain is intended to isolate the root link from the rest of the network so that potential problems on the network cannot disrupt the tests.

The test itself consists of a program which has a header and two node transactions, both directed to the root node. The header causes a poll to be conducted. The first transaction commands the node to disable all of its ports. The second transaction commands the root node to enable its root port only once, for the response to this command. The program concludes by moving a command to the ICR to cause a transition from solicited to unsolicited mode.

If the interface status of the IOS under test is still idle (i.e. other tests have not detected any errors) and GPC FDIR has not detected any faults in the channel to which the IOS is connected, the test will proceed. The starting address of the test program is written to the solicited chain pointer and the ICR is commanded to start executing a solicited chain. This process then waits for the chain to complete; a delay of ten milliseconds is adequate.

After the delay, IOS status registers and the transaction status and data are read and stored in FTP RAM for further analysis. A second check for the health of the FTP channel is made. If the channel connected to the IOS failed during the test, no further analysis is performed and the interface status is marked Failed_Channel. Otherwise, the analysis proceeds starting with the IOS status registers, followed by the transaction status and finally the data returned by the root node. The IOS status registers are scanned to see if any errors were detected which would result in the interface status of the IOS being marked Failed_IOS. When an error is detected, it is logged to the Network Error Log and the rest of the analysis is not performed. First, the value of the CSR is analyzed. The chain complete bit should be set. If it is not, the IOS is marked Failed_IOS since this error implies either an IOS which cannot switch modes (from solicited to unsolicited), a problem with the TLR in aborting a solicited input command which has timed out, or a root node which cannot be disabled and is allowing a coherent babbler to be heard by the IOS. The other fields of the CSR should hold their reset values. The ISR is examined to detect a stuck on high condition of the network bus, another Failed_IOS condition. The value of the CSR prior to the mode switch is called the final CSR. It also has error detection information. Its value should reflect: network possession by this GPC, no possession default (a poll detected during this chain execution), no data transmission faults (an incoming message detected during a message transmission) and no poll transmission faults (data transmission detected during the conduct of the poll). Errors detected here result in the network interface status being marked Failed _IOS.

After analyzing the IOS status registers, status and data from the two node transactions are analyzed. The status is appended by the IOS when the solicited input instruction is executed; the data is the message returned by the node. The first test transaction commanded the node to disable all its ports. Thus, no reply to this transaction should have been received. The instructions controlling this transaction programmed the TLR to move to the next instruction if no input is received within 512 microseconds. Proper execution of the solicited input instruction involves zeroing the byte count field of the transaction status. A non-zero value is written to this field as part of the IOS/DPM initialization. A delay of approximately 512 microseconds (measured by reading and saving the time register immediately before and after the solicited input instruction) and a zero byte count is evidence that these IOS operations are functioning properly. Also the status of the HDLC device is stored after the outgoing message is transmitted. It should indicate no transmission errors were detected.

The second test transaction commands the node to enable its root port for its response to this transaction only. (The node is left with all ports disabled until the network manager begins execution.) Each status field is analyzed for error indicators. The HDLC SR should indicate no transmission errors after sending the node command. The byte count field should have a value of $0F_{16}$. The HDLC IR should indicate no HDLC protocol errors were detected while receiving the node response. The HDLC SR should indicate that the message from the node had three residual bits. The checksum is validated by calling a subprogram from IOS Utilities. Finally, the address transmitted by the root node is verified as being correct.

3.2.4 IOS Utilities For I/O Network Managers

I/O Network Managers have two basic objectives when communicating with the nodes in the network: status collection and reconfiguration. During status collection, each node in the network is commanded to send back the data which has been stored in the node's status registers. Reconfiguration is the process by which a node is given the particular port enable pattern it must maintain until the next reconfiguration command is received. In order to achieve these objectives, the I/O Network Manager must have access to the services of the IOS since both status collection and reconfiguration are accomplished by means of IOS programs, also referred to as chains. This process provides the I/O Network Manager with the capability to execute and manage chains on an I/O network. It provides this service through a simple interface which conceals all of the details of the IOS hardware interface from the Network Manager.

Status collection is the primary means by which a Network Manager determines whether or not any faults are present in the network. Although the data returned by the nodes is useful in the analysis of errors, the absence of an expected response and the condition of the bus itself are also very useful diagnostic tools. Collecting node status, i.e. eliciting a response from each non-failed node in the network, is accomplished by means of a status collection program or chain. This chain does not change unless a node failure is detected. The chain is fixed or static in two ways. The program executed by the IOS when running the chain does not change and the data sent to each node during the chain execution does not change. When a node is failed, it is no longer queried for its status. A failed node is isolated from the network, hence it will not even receive the command asking for its status. Therefore, the transaction which was initially sent to the node is removed from the chain. This process is called transaction deselection and is accomplished by modifying the status collection program. Similarly, if a node is brought back on line, the transaction which collects its status is returned to the status collection chain or selected again. To support status collection, this process provides the Network Manager with the ability to run a status collection chain, to select a status transaction, to deselect a status transaction, and to update a status chain in the IOS of a recovered FTP channel.

Reconfiguration chains are not static because the number of transactions in the chain varies with the type of reconfiguration the manager is trying to effect. Furthermore, the data sent to the nodes which are being reconfigured is intrinsically dynamic: the configuration of each node is determined as a network is grown or repaired and is not known a priori. This process provides the Network Manager with the capability to execute reconfiguration chains where both the number of transactions and the data sent to each node is specified by the manager at the time the service is called.

While carrying out its principle function of sending and receiving data, the IOS is designed to detect various error conditions on the network. Some detectable error conditions concern the network as a whole, some concern the IOS, and some concern the individual node transactions. Whenever a chain is executed for an I/O Network Manager, this information will be analyzed to the extent possible. This analysis in turn is returned to the Network Manager who uses it as the basis of further analysis and of its reconfiguration strategy.

In general, if this process detects an error when executing chains for the Network Manager, it will log the error and any pertinent error information in the I/O Network Error Log. However, the Network Manager may execute chains containing transactions intended to produce an error symptom. Thus, this process will give the Network Manager the option of conducting chains without logging detected errors. Finally, this process will provide the Network Manager a means of testing the network for the presence of a babbler without having to execute either a status collection or reconfiguration chain.

3.2.4.1 Process Name:	Execution of Node Reconfiguration Chains
Inputs:	I/O Network Identifier Active Root Link Configuration Commands Contention Option Logging Enable
Outputs:	Configuration Report
Requirements Reference:	I/O Network Functional Requirements, Section 2.3.1, 2.4.2
Notes:	None.

Description:

This process is responsible for executing chains for the Network Manager of the network specified by the I/O Network Identifier. It is assumed that the purpose of these chains is to reconfigure the network, however, the Network Manager may use this process to conduct I/O transactions for any purpose it deems necessary. These chains differ from other I/O chains in the system in two respects: the number of transactions in the chain is not constant and the chain may not always be conducted with contention. The number of transactions in the chain depends upon the type of reconfiguration the Network Manager is trying to effect. For example, if the root node is being reconfigured, only one transaction will be in the chain. If a link between two nodes is being enabled, then two transactions will be in the chain. The number of transactions is bounded only by the number of nodes in the network. The Network Manager has the option of running chains either with or without contention for the network. In general, contention is used even for chains run on local or dedicated networks, where contention for the use of the network is not necessary, since some valuable diagnostic information about the state of the network can be obtained in this way. However, when certain fault conditions are present in the network, it may not be possible to win a contention. Thus, to force the chain to be executed, the Network Manager can request that the chain be executed without contention. This process is responsible for conducting the chain as specified by the Manager in the Contention Option.

In order to invoke this process, the Network Manager must supply an ordered list of commands to be sent to nodes in the network. These Configuration Commands contain data in the proper format which is to be sent to the nodes as part of this chain.

This process is divided into two main functions: chain execution and error analysis. The result of the error analysis is returned to the Network Manager as a Configuration Report. The Configuration Report informs the Manager about the outcome of the attempt to execute this chain. The report takes the form of a discriminated record. One field of the record indicates whether or not an interface failure was detected when executing this chain, and if there has been, whether it is due to a failed FTP channel or failed IOS hardware. When no interface failure is detected, another field indicates whether or not a babbler has been detected during the execution of this chain, and if there has been, whether not it was detected during contention for the network or during data transmission. When neither a failed interface nor a babbler is detected, the last field contains status and data resulting from the conduct of each transaction in the chain. The status information states whether or not any communication protocol errors were detected during this transaction. When no errors are detected, the data returned by the node is also returned to the Network Manager.

Chain execution begins by checking with GPC FDIR as to the status of the FTP channel to which the Active Root Link is connected. If the channel is okay, chain execution will continue, otherwise the report returned to the Manager indicates a channel failure.

Next, the program which will execute this chain is tailored to meet the requirements of this chain as specified by the Network Manager. It is by this means that the Manager's perogatives over contention and the number of transactions are implemented. First, the IOS instruction which controls the type of contention used for this chain is generated. If the Contention Option asks for no contention, the instruction will simply be given a value of No-op. If the Contention Option indicates contention is required and the network is local (i.e. dedicated to use by one GPC), an instruction is generated which moves a command to the ICR to cause a poll to start immediately. In the case where a network is regional and contention is required, the command indicates a normal polling sequence is to be used. Different polling sequences are used because the immediate poll takes less time to complete than a full polling sequence. Polls are conducted on local networks for error detection only and therefore a performance gain is obtained by using the immediate poll. However, on regional networks the full contention protocol must be followed since the poll also determines which GPC will have control of the I/O network. The second step in tailoring the program is to write the address of the end of chain program to the operand field of the branch instruction of the last transaction in this chain. The original value of the operand field is saved so that it can be restored after the chain is complete. In this way, the number of transactions in the chain can be varied from a minimum of one to a maximum equal to the number of nodes in the network.

Next, the input and output records for this chain are initialized. The output records receive the voted value of the commands as given by the Network Manager. A non-zero bit pattern is written to the fields of the input record.

To cause the IOS to execute the chain, the address of this chain is written to the Solicited Chain Pointer and the command to start the chain is written to the ICR. This process then waits for a specified amount of time before proceeding. The amount of time depends on the number of transactions in the chain. Presently, the value of the timeout is one millisecond per transaction plus one additional millisecond for miscellaneous overhead (such as the time to complete a contention). The wait may be accomplished as a busy wait or as a request to the operating system to suspend this process. The busy wait alternative should be chosen when the value of the timeout is close to the amount of time required to perform a process switch. In this case, the system does not perform useful work while this process is suspended; however, chain completion will be detected immediately thus providing the manager with a performance gain. The final step in chain execution is to read the data and status information produced by the chain from the IOS/DPM into local FTP memory using the implicit data exchange mechanism of the LMN region to maintain source congruency in all channels.

The second part of this process is error analysis. This begins by verifying that the channel connected to the IOS conducting this chain has not failed during chain execution. There are two parts to this diagnostic procedure: a data exchange pattern test and a call to GPC FDIR. Since GPC FDIR is a periodic process, a small amount of time may elapse between the failure of a channel and its detection by FDIR. The data exchange pattern test is used to detect a failed channel which GPC FDIR has not yet uncovered. If the channel with the active root link has failed, non-failed channels will obtain invalid data from its IOS/DPM. This data should not be processed since it could result in erroneous conclusions about the network. Similarly, if the channel failed after the last check with GPC FDIR (before the chain data was loaded into the DPM) but the failed channel has been resynchronized by GPC FDIR, then the data exchange pattern test will show no errors but again the chain data should not be processed since it may be invalid. To prevent this situation from occurring, a call is made to GPC FDIR. When a channel has failed and then been restored, GPC FDIR will not report its status as okay until it has undergone a trial period in a resynchronized state. This period is much longer than the longest chain delay. This means that errors resulting from a channel which failed before voted data was written to the DPM and which is now functionally resynchronized are still correctly attributed to the failed channel. The way in which checks are performed on the status of the channel which interfaces to the active IOS creates a window of time during which it is possible to determine whether or not the channel has failed. This test is important because use of invalid data could result in erroneous conclusions being drawn about failures in the network. Thus, this process is protected from using invalid data due to a failed channel. If the channel is okay, the error analysis will proceed, otherwise a log entry is made and a report indicating a failed channel is returned to the Network Manager.

The need to have this window of time during which a channel failure is known not to have occurred also drives the sequence of steps followed in reading and analyzing the data from the IOS/DPM. Thus all the data is read from the IOS/DPM before error processing is started, rather than a sequence where part of the data is read and analyzed and if no errors are detected, more data is read. After each section of code used to transfer data from the IOS/DPM to local memory, it is necessary to verify that the channel connected to the IOS has not failed. Since errors are relatively infrequent occurrences, the simplicity and speed of the read and process method is preferred even though it may occasionally result in reading data that is later discarded because detection of errors has made it suspect. This sequence provides the greatest performance benefit for the most common behavior of the system. It should also be noted that log entries are only made by this process if the value of the Logging Enabled parameter indicates that log entries are to be made whenever errors are detected.

When no channel failures are detected, error analysis proceeds on the data which was copied from the IOS/DPM and exchanged across all channels of the FTP. If the value of the Chain Status Register (CSR) indicates that the chain did not complete in the allotted time, a command is written to the Interface Command register (ICR) to stop the IOS in case it is still executing a chain or has failed in such a way that is is in an infinite loop and possibly babbling on the network. A check is then made of other error indicators to determine if an incoming babbler was detected or if the IOS has failed. The indicators that are examined are the contention state of the IOS and the possession default indicator in the CSR if the chain was executed with contention, the extent to which the chain did complete as indicated by the value of the solicited chain pointer, the extent to which the IOS correctly performed its byte count zeroing function when executing a receive input instruction, and the ability of the DPM to pass a read/write pattern test. When a chain does not complete, the CSR is not reset. Therefore, the value used in the above analysis is the value last read from the CSR. If errors are detected, a log entry is made and the type of error is returned to the Network Manager. Otherwise, the error analysis will proceed with a check for a babbler condition if the chain did complete.

When a chain completes, the value in the CSR is reset, thus the analysis to determine whether or not a babbler is present in the network is performed on the final value of the CSR which is saved by the end of chain program prior to commanding the ICR to switch modes (the definition of chain completion is a switch from the solicited to the unsolicited mode of operation). The final value of the CSR is examined for an indication of data transmission on the network while an output instruction is being executed by this IOS. Furthermore, if the chain is conducted with contention, the final CSR is examined for indications that data was transmitted on the network during the polling sequence or that a polling sequence was attempted during data transmission by this IOS. Any of these three protocol violations are assumed to be evidence of a babbler on the network. If any of these errors are detected, a read/write pattern test is performed on the DPM to ensure that the error is due to a babbler and not a failed DPM. If the results of this analysis indicate the detection of an error, a log entry is made and the error type is returned to the Network Manager. Otherwise, the error analysis will proceed to examine the data in the Interface Status Register (ISR) and to verify that the CSR has been reset.

If the CSR has not been reset, the error is logged and the error report to the Network Manager will indicate a failed IOS. Otherwise, the ISR is examined for the presence of a stuck-on-high condition of the network. If this condition is detected, an entry is made to the error log and the report returned to the Network Manager indicates a failed IOS. Otherwise, the status information from each node transaction is analyzed.

The status information from each node transaction is examined for error information as follows. The HDLC status, which is saved after the transmission of the command to the node, indicates whether or not any framing or overrun errors occurred during the transmission. If this error is detected, the IOS is considered failed; a log entry is made and the report to the Network Manager indicates a failed IOS. If the byte count kept by the IOS on the data returned by the node still has its initial (non-zero) value, the IOS is considered failed. This value should be a value from zero to fifteen. Fifteen is the correct byte count, zero indicates no response is received from this node and any value in between is an incomplete transmission from the node. The IOS when operating correctly will zero this byte count and then start to increment it as data is received from the node. When the initial value has not been written over by the IOS, it is assumed that the IOS is not operating correctly. This error results in a log entry being made and the report to the Network Manager indicates a failed IOS condition was detected.

In the cases where the error is attributed to a failed IOS, no further error processing is performed. However, some errors are attributed not to the IOS but instead to the transaction, i.e. the node, whose status is being analyzed. When errors attributable to a node are detected, the error analysis proceeds to examine the status of the remaining transactions in the chain. Thus, if the byte count has any other value except the correct byte count of fifteen, the error is attributed to the transaction itself and not to the IOS. In particular, if the byte count is zero, then no response was received from this node. This error condition is logged and the report returned to the Network Manager will indicate that the transaction to this node had an error. The status of each transaction is then examined for the presence of HDLC protocol errors, the transmission by the node of an incorrect number of residual bits and an invalid sumcheck appended to the message. The detection of any of these errors results in a log entry being made and an error indication being scored against that transaction. Should any of these errors be detected, a read/write pattern test is performed on the DPM to be sure that the error is not attributable to a failed DPM memory. A failed memory results in a an error report to the Network Manager indicating a failed IOS. If the memory test indicates that the memory is okay, the error report will indicate which individual transactions had an error. When a transaction has no errors scored against it, the data associated with that transaction is returned by this process to the Network Manager as part of the final report. However, if the transaction has errors scored against it, no data from that transaction is returned to the Network Manager. When all the transactions have been subjected to this error analysis, this process is complete.

3.2.4.2 Process Name:	Execution of Node Status Collection Chains
Inputs:	I/O Network Identifier Active Root Link Logging Enable
Outputs:	Status Collection Report
Requirements Reference:	I/O Network Functional Requirements, Section 2.3.1, 2.4.2
Notes:	None.

Description:

This process is responsible for executing status collection chains for the Network Manager of the network specified by the I/O Network Identifier. Unlike reconfiguration chains, these chains resemble other I/O chains in the system in that the number of transactions in the chain is constant and the chain is always conducted with contention. The differences between the reconfiguration chain and the status collection chain necessitates some differences in the way in which these chains are executed. However, the differences are few in number and will be described here.

Since the status collection chain is always run with contention, the Network Manager does not need to supply this process with a Contention Option. Similarly, since the data which is used to collect status from the nodes is static, the Network Manager does not need to supply a list of commands to be sent to the nodes. It does, however, expect to receive a report back from this process. The Status Collection Report is a summary of the analysis of status information obtained by the IOS during the execution of this chain. The report takes the form of a discriminated record. One field of the record indicates whether or not an interface failure was detected when executing this chain, and if there has been, whether it is due to a failed FTP channel or failed IOS hardware. When no interface failure is detected, another field indicates whether or not a babbler has been detected during the execution of this chain, and if there has been, whether not it was detected during contention for the network or during data transmission. When neither a failed interface nor a babbler is detected, the last field contains status and data resulting from the conduct of each transaction in the chain. This information is also packaged as a discriminated record, one for each transaction in the chain. The first field of this record indicates whether or not the transaction is selected. When the transaction is selected, another field contains the status of the transaction. If the transaction had no errors, the last field will contain the data returned by the node.

This process is divided into two main functions, chain execution and error analysis, in the same manner as the process which conducts the reconfiguration chain. The execution of the status collection chain differs from the execution of the reconfiguration chain only in that no changes to the program which executes this chain are made and no output records need to be initialized. Furthermore, once the command to start the chain is written to the ICR, the process suspends itself while waiting for chain execution to complete. The process suspends itself for the number of milliseconds equal to the number of selected transactions in the chain plus one.

The error analysis is identical to that of the error analysis conducted for the reconfiguration chain with one distinction: if the transaction is deselected, no processing is done on that transaction. The report returned to the Network Manager also indicates that this transaction is deselected. Since this process may be called by the Network Manager when errors are expected, i.e. when no response is the valid response to a transaction to a node, the option of disabling error logging is also provided by this process.

3.2.4.3 Process Name:	Management of Status Collection Transactions
Inputs:	I/O Network Identifier Node Number Active Channel
Outputs:	Updated Status Chain
Requirements Reference:	I/O Network Functional Requirements, Section 2.4.1
Notes:	None.

Description:

This process is responsible for maintaining the node status collection program in the state requested by the Network Manager and for maintaining a history of this information. In this context, there are three services which this process provides: transaction deselection, transaction selection, and updating the status collection chain in an IOS connected to a recovered FTP channel.

The node status chain is an ordered set of transactions, one to each node in an I/O Network. The program which controls chain execution consists of a set of instructions called a header, which is executed once per chain at the start of the chain, and a set of repeated instructions, one for each node in the network, which controls the transmission of data on the network. Each member of this set is identical to every other member with a few exceptions. The differences are the address of the output packet in the output instruction, the address of the input packet in the receive input instruction, and the address of the next instruction in the branch instruction. Transaction selection simply manipulates the address operand of the branch instruction so as to either bypass or include the transmission of a status command to a node.

To deselect a transaction, the Network Manager calls the subprogram Deselect Node Status Transaction with the Network Identifier of its network, the Node Number of the node whose status transaction is to be bypassed, and the channel identifier of the FTP channel containing the active IOS. If the transaction is already bypassed, no action takes place. However, if the transaction is not already bypassed, the deselection is logged in a static variable, called the Selection Status, which is maintained by this process. Each network has its own Selection Status with an entry for each node in the network. Another static variable, called the Active Node Count, is decremented. Next a read pointer and a write pointer are assigned values. The read pointer is assigned a value which allows it to read only from the active DPM while the write pointer will write to all the DPMs connected to this network. The Selection Status of this network is then searched in reverse order starting with the transaction which is about to be bypassed for the selected node closest to (but preceding) that transaction. This is the node whose status is collected before the status of the node which is undergoing deselection. If such a node is found, the address operand of its branch instruction is changed to point to the instruction pointed to by the branch instruction of the deselected transaction. If a selected node is not found, as would be the case if the node being deselected is the first node in the chain, the address operand of the branch instruction in the chain header is changed to point to the instruction pointed to by the branch instruction of the deselected transaction. The read and write pointers are used so that whenever a value is read form the DPM, as with the value of the address operand of the branch instruction of the deselected transaction, it is read from one channel only and exchanged across all channels. However, whenever a value is written to the DPM, it is written to all the DPMs connected to this network simultaneously. The former is necessary because channel failures or simplex operation of the system will result in errors read from multiple, noncongruent sources. The latter is necessary so that DPMs connected to all non-failed FTP channels have the correct, current version of the status collection chain.

To select a transaction, the Network Manager calls the subprogram Select Node Status Transaction with the Network Identifier of its network, the Node Number of the node whose status transaction is to be bypassed, and the channel identifier of the FTP channel containing the active IOS. If the transaction is already selected, no action takes place. However, if the transaction is currently bypassed, the read and write pointers are generated as for the transaction deselection process. Again, the Selection Status is searched in reverse order starting with the transaction which is to be selected for the non-bypassed transaction closest to (but preceding) that transaction. If such a transaction is found, the address operand of the branch instruction of the transaction being selected is changed to point to the same address that the preceding transaction points to. Following that change, the address operand of the branch instruction of the preceding transaction is changed to point to the transaction being selected. When no preceding transaction is found, as would be the case if the transaction being selected is the first transaction in the status collection chain, the address operand of the branch instruction of the transaction being selected is changed to point to the same address that the branch instruction in the header points to. Following that change, the address operand of the branch instruction of the header is changed to point to the transaction being selected. The read and write pointers are used in the same fashion as in the transaction deselection procedure. Finally, the Selection Status of the selected transaction is updated to indicate that this transaction is selected and the Active Node Count is incremented.

The third subprogram provided by this process is called Update Node Status Chain. It is used whenever an FTP channel has become desynchronized due to a transient fault and therefore taken offline by the GPC FDIR process but later restored to service. Any changes made to the node status chains of networks connected to this GPC through the failed channel will not actually reach the DPMs in that channel. Of course, the changes will be made to the node status chains residing in DPMs connected to non-failed channels. When the failed channel is brought back on line and all its local memory has been aligned, its DPMs must be reinitialized. Part of the reinitialization process will be to update the status collection chains in these DPMs. The process begins by initializing a pointer which points to one channel only, i.e. the channel being restored, since the changes made to this chain already exist in DPMs belonging to other root links in the network. Next, the address operand of the branch instruction in the header is initialized to point to the end of chain program. Each value in the Selection Status of this network is then examined in turn starting with the first transaction in the chain to determine whether or not that transaction is selected. If it is not selected, it is simply passed over. If it is selected, however, the address operand of the last selected transaction is changed to point to this transaction. In the case of the first selected transaction that is found, the operand of the branch instruction in the header is changed instead. When the last selected transaction is found, the address operand of its branch instruction is assigned the address of the end of chain program.

3.2.4.4 Process Name: Testing for Presence of Babbler on Network

Inputs: I/O Network Identifier
Active Root Link

Outputs: Babbler Report

Requirements Reference: I/O Network Functional Requirements,
Section 2.4.1, 2.4.3

Notes: None.

Description:

This process is responsible for executing a chain on the network which will detect the presence of a babbler. This subprogram is called by the Network Manager as a diagnostic tool when it suspects the presence of a babbler on the network. The test is accomplished by using the header of the reconfiguration program as if it were the entire program. As with status collection and node reconfiguration, the processing is performed within a window of time during which channel failures of the active root link can be detected. The address operand of the branch instruction of the header is changed to point to the end of chain program and restored at the completion of the test. The process suspends itself while waiting for the chain to complete. To extend the time over which babbler detection can occur, the full poll, rather than an immediate poll is conducted. The various fields of the CSR, the final CSR and the ISR are examined for evidence of a babbler on the network. This processing is described in detail in section 3.2.4.1. If a babbler, a failed channel, or failed IOS is detected during this diagnostic test, the error is logged and a report is returned to the Network Manager indicating the type of error which was detected. Otherwise, a report is returned to the Network Manager indicating that no errors were detected.

3.3 I/O Network Databases

The I/O Network Databases serve as a repository of static information about I/O networks. They contain a software description of the physical makeup of the I/O networks in the system. They also contain the information necessary to map logical data related to networks into physical data. The databases also contain information about the organization of the I/O networks into I/O Services.

The I/O Central Database holds information about every I/O network and every I/O Service in the system. The I/O Local Database contains information about the I/O networks to which a particular GPC is physically connected. The I/O Local Database references the I/O Central Database during program initialization to obtain information about the networks to which its GPC is connected. Using this information, the I/O Local Database deduces other information about its networks and stores all this data locally.

3.3.1 Process Name:	I/O Central Database
Inputs:	I/O Network Identifier I/O Service Identifier GPC Identifier
Outputs:	Network Topology I/O Service Descriptor Connection Indicator
Requirements Reference:	I/O Network Functional Requirements, Section 2.3.2
Notes:	In systems with mass memory, this data is stored on those devices. In systems without mass storage, each GPC will have an instance of this process.

Description:

This process is responsible for providing users with accurate, consistent information about the physical makeup and logical organization of all the I/O networks in the system. This information can be stored as binary data in a file or it can be generated by assignment statements in a program and then transferred to an appropriate storage medium. In either case, prior to accepting calls from users, it will verify that the network topologies it can provide are self consistent. This means that node to node connections are commutative, i.e. if node 1 is connected to node 2, then node 2 is connected to node 3. Furthermore, since a network may only be assigned to only one I/O Service, it will verify that a network identifier appears in only one I/O Service descriptor. Since this database is static, these checks need to be performed only once. If any errors are detected in the I/O Central Database, an error message will be displayed for an operator and further initialization will be aborted. In this way any possible run time problems due to faulty data are eliminated.

To obtain the topology of a network, a user must provide this process with the logical identifier of that network. To obtain a service descriptor of an I/O Service, a user must provide this process with the logical identifier of that service. To obtain a network connection indicator, a user must provide this process with the logical identifier of the GPC about which it wants network connection information.

The primary users of this process are the I/O Local Databases from the various GPCs in the system. Other users could be the Resource Allocator and the System Manager.

3.3.2 Process Name:	I/O Local Database
Inputs:	I/O Service Identifier I/O Network Identifier Channel Number
Outputs:	Connected Networks Available I/O Services Network Topology I/O Service Descriptor Root Links Channel Identifier
Requirements Reference:	I/O Network Functional Requirements, Section 2.3.2
Notes:	None.

Description:

This process is responsible for providing users with information about networks to which a GPC is physically connected. This information is assembled at initialization or power up time for use by other processes during run time. During system initialization, this process reads data from the I/O Central Database about the I/O networks to which it is connected. Its first action is to obtain from the central database a list of networks to which it is connected and a list of I/O Services which it must support. For each network to which it is connected, it obtains from the central database a copy of that network's topology. For each of these networks, it uses the information in the topology definition to generate a list of which connect this network to its GPC. Rootlinks in an I/O network must meet certain criteria as stated in Appendix B, Network Operating Rules. Once the data describing the rootlinks has been collected, it is reviewed for correctness in accordance with these rules. If any errors are detected in the I/O Local Database, an error message will be displayed for an operator and further elaboration will be aborted. As with the I/O Central Database, this eliminates any possible run time problems due to faulty data.

To obtain the list of networks to which this GPC is connected, a user makes a call to a function which needs no parameters but which returns the list of connected networks. A similar call will return to a user a list of I/O Services which are available to this GPC. To obtain the topology of a given network, a user must provide this process with the logical identifier of that network. In a similar way, a user can obtain information which describes the rootlinks of a given network. To obtain the I/O Service Descriptor of an available I/O Service, a user calls a function with the logical identifier of that service. Finally, the physical identifier of the channel containing a given rootlink is returned by a function which has been provided with the logical identifier of that channel.

The primary users of this process are Network Managers, I/O Communication Manager, and I/O Network Status.

3.4 I/O Network Status

I/O Network Status serves as a repository of information about the state of every network in the system. Furthermore, since the network is a physical resource under software control, the state is also comprised of information about the logical process which has access to the network at any given time. Two processes share responsibility for determining network status: the I/O Network Manager and the I/O Communication Manager.

The hardware components in the network which are viewed as part of the AIPS system are the nodes, the ports of the node, and the IOSs. (A link is defined as two ports on adjacent nodes and the cable between them.) The state of the nodes and the IOSs is determined solely by the Network Manager. This information is stored in Network Hardware Status. Thus the hardware status is the Network Manager's view of the network hardware made available to any other process in the system. Of course, the actual physical state of the hardware may change many times during network growth and reconfiguration. However, these transitionary periods are of short duration. Thus the values stored in Network Hardware Status are stable values representing the view of the Network Manager after any necessary changes in configuration have been made. The state of DIUs, the rootlink currently in use, and who controls access to network resources is determined jointly by the I/O Communication Manager and the I/O Network Manager. This information is stored in Logical Network Status.

Since there may be several GPCs in a system, the status for a given network resides initially on the GPC which hosts the Network Manager of that network. A process on one GPC which needs to obtain the status of a network connected to another GPC will use Intercomputer System Services to effect the transfer.

3.4.1 Process Name:	Network Hardware Status
Inputs:	I/O Network Identifier Node Status Interface Status Network Status Network Is Active Flag Update Generation
Outputs:	Interface Status Network Status Update Generation Updated Status Flag
Requirements	I/O Network Functional Requirements,
Reference:	Section 2.3.3
Notes:	None.

Description:

This process is responsible for maintaining current information about the status of the network hardware of all the I/O networks in the system. Network hardware consists of nodes, links and IOSs. An instance of this process exists on every GPC which is connected to an I/O network. For each of these networks, a status object is allocated and initialized. Any necessary information about the number of nodes and IOSs in the network is obtained from the I/O Local Database. Since several processes may need to access this data, each status object must be protected so that the information it contains is consistent. That is, read/write accesses to the status object are restricted so that only one outside process may write to the object at a time and that during this operation no other processes can read the object. Furthermore, during a read operation, no outside process can write to the object. However, several processes may simultaneously read the object.

The status of a node may be active, failed, or idle. The initial value of node status is idle; however, nodes may have an idle status only prior to the activation of a Network Manager for the network. Once the Network Manager has initialized the network hardware, the status of a node must either be active or failed. When a node is failed, it means that no operational link to that node exists. When a node is active, it means that the node has an operational link to the rest of the network and that the Network Manager detects no protocol violations when communicating with the node.

Links consist of two ports, one on each node, and the wire that connects them. Since it is not possible to isolate a link failure to one of these three components, link status in fact is described by the port status of the two ports which are at opposite ends of the link and which always therefore have identical status. The status of a port is either idle, active or failed. Unlike nodes, a port may.continue to have an idle status after a network is initialized by the Network Manager. When the status of a port is idle, it means the port hardware is not currently enabled. However, there are two possible reasons for this status. First, there may be no other network element connected to that port. Second, the adjacent element is a node and this port is part of a spare link to that node. As a spare port, it could be used by the Network Manager to reconfigure the network in the event of a failure somewhere else in the network. When a port is idle, it does not pass along messages which its node receives on its other ports, and an idle port does not transmit (to other ports of its node) messages which it receives. When the status of a port is active, it means the port communication hardware is enabled. An enabled port gates messages it receives to other enabled ports in its node and it retransmits messages received by other enabled ports to other elements in the network. When a port is failed, it means that the Network Manager has detected some communication protocol violation when using this port. The enable/disable state of the hardware is not necessarily known. The failure may actually exist beyond the port itself. What is significant about this status is that it indicates the boundary of a fault containment region. The Network Manager will not try to use a failed port as part of the network.

The status of an IOS is either idle, available, active, failed-channel or failed-IOS. The initial value of the status of an IOS is idle. During the power on sequence each IOS and its root node are given a series of diagnostic tests. If they do not pass these tests, the IOS status is downgraded to either failed- channel or failed-ios, depending on the cause of the failure. In a manner similar to that of nodes, IOSs may have an idle status only prior to the activation of a Network Manager for the network. Once the Network Manager has initialized the network hardware, the status of an IOS must be one of the other four allowable values. An IOS with a status of failed-ios has been diagnosed by the Network Manager or the power on test sequence as having a serious hardware fault. Such a fault can be detected while using the IOS to run chains on the network. An IOS with this status is no longer used by the Network Manager or by the I/O Communication Manager for any network access. An IOS whose status is failed-channel, however, may not have any hardware faults of its own but, nevertheless, cannot be used to execute chains because it is connected to a channel which, according to GPC FDIR, has failed. If the channel is brought back online, the status of this IOS is upgraded to available. Either type of failed IOS status is considered a root link failure. An IOS whose status is active is the IOS which is currently being used to execute chains An IOS whose status is available is ready to become the active IOS if there is a root link failure of any type in the active IOS.

This process supports several types of read/write operations. Network status consists of node status and interface status. It is possible to read or write either part separately or both parts in one subprogram call. The object here is efficiency: if the status of only one part has changed, the writing process can indicate to this process which part to update. This process will only update that part. If a user is only interested in one part of the status, it may read only that part. Furthermore, a user can determine whether or not any part of the status has changed since the last time it read status and therefore avoid an unnecessary transfer of data. There is no reason to read status if the copy is already current.

Finally, in a distributed AIPS System it will be necessary to route calls for status to the proper GPC for reply. This will require use of Intercomputer System Services.

3.4.2 Process Name: Logical Status

Inputs:
I/O Network Identifier
Network Usability
Current Root Link
Unreachable DIUs

Outputs:
Network Usability
Current Root Link
Unreachable DIUs

Requirements Reference: I/O Network Functional Requirements, Section 2.3.3

Notes: None.

Description:

This process supports the protocol which governs access to an I/O Network. The two processes which may access a network are the I/O Network Manager of that network and the I/O Communication Manager. Since only one of these processes can use a network at any given time, a protocol needs to be in place to ensure non-overlapping use of the network by these processes. Additional information about the status of the network is also managed by this process. This includes the list of Unreachable DIUs and the root link being used by the process in control of the network.

Network Usability may have one of three values, in-service, out-of-service, and repaired. Whenever the network is in-service, it may only be accessed by the I/O Communication Manager. Furthermore, only the I/O Communication Manager may take a network out-of-service. Once out-of-service, however, a network may only be accessed by its Network Manager. The Network Manager can set Network Usability to repaired, and the I/O Communication Manager in turn can put a network back in-service. Whenever the I/O Communication Manager detects a communication error when using a network, it takes the network out-of-service by calling a subprogram in this process. The I/O Network Manager can then be activated to perform its FDIR on the network. Once it has reconfigured the network, it determines which, if any, DIUs are now unreachable, i.e. cannot communicate over the network due to known hardware faults in the network. A DIU is unreachable if the node to which it is attached is failed or if the port adjacent to the DIU is failed. This latter case means that during the growth of the network, when an attempt was made to enable this port, it appeared that the DIU was babbling and therefore needed to be isolated from the network. After recording this list of unreachable DIUs, the Network Manager indicates that it will no longer use the network by setting the Network Usability to repaired. The I/O

Communication Manager can then put the network back in-service. Whoever has access to the network, also has write access to the shared variables of this process. Thus, the value of the current root link is the value being used by whichever process controls the network. The controlling process may change this to any other available root link if necessary. When control is transitioned from one process to the other, the incoming process adjusts whatever local data necessary to be able to make use of the current root link.

3.5 I/O Network Logs

This process is responsible for keeping a log relating to the history of network hardware for each network in the system. The Network Manager and the I/O Communication Manager both make log entries. The entries can be displayed on a terminal.

3.5.1 Process Name:	I/O Error Logs
Inputs:	I/O Network Identifier Log Entry
Outputs:	Display of Log Entries
Requirements Reference:	I/O Network Functional Requirements, Section 2.3.4
Notes:	None.

Description:

This process is responsible for maintaining a circular log of error information for each network in the system. The errors will be recorded by the Network Manager and the I/O Communication Manager. An error is loosely defined as any communications protocol violation detected by any software module in the above processes. The logs are therefore general purpose. The log should have fields for the network identifier, a node or root link identifier, a string which identifies the subprogram making the log entry, a string to describe the error and a field to allow up to four bytes of hexadecimal data to be recorded in the log. Since all entries may not need all possible fields, various combinations of the above fields should be allowed. Log entries will be time stamped by this process.

The log will display its entries in a chronological fashion, with the oldest entries appearing first on the display window. Each displayed entry will show the time stamp to the nearest ten milliseconds, the string identifying the caller, and all other data stored in the entry in neat columns. It will be possible to display the log for any network in the system.

3.5.2 Process Name: I/O Event Logs

Inputs: I/O Network Identifier
Log Entry

Outputs: Display of Log Entries

Requirements Reference: I/O Network Functional Requirements, Section 2.3.4

Notes: None.

Description:

This process is responsible for maintaining a circular log of event information for each network in the system. The events will be recorded by the Network Manager and the I/O Communication Manager. An event is loosely defined as any occurrence of interest which any software module in the above processes wishes to record. The logs are therefore general purpose. The log should have fields for the network identifier, a node or root link identifier, a string which identifies the subprogram making the log entry, a string to describe the error and a field to allow up to four bytes of hexadecimal data to be recorded in the log. Since all entries may not need all possible fields, various combinations of the above fields should be allowed. Log entries will be time stamped by this process.

The log will display its entries in a chronological fashion, with the oldest entries appearing first on the display window. Each displayed entry will show the time stamp to the nearest ten milliseconds, the string identifying the caller, and all other data stored in the entry in neat columns. It will be possible to display the log for any network in the system.

3.6 Process Name: Network Status Monitor

Inputs: I/O Network Identifier
Network Display Database
Network Status

Outputs: Network Display

Requirements Reference: I/O Network Functional Requirements, Section 2.3.5

Notes: None.

Description:

At present displays for three network topologies are available, a six node network, a ten node network, and a fifteen node network. Each display is run as part of the local operating system in each GPC. Figure 20 is a diagram of the VT100 (black and white) display of a 15 node network. The node ID is in the middle of each node with the five port IDs surrounding it. The display uses the three levels of intensity of the VT100 graphics to distinguish failed nodes, links or ports from active nodes, links or ports. Dashed lines indicate idle links, solid lines indicate active links, and dashed, highlighted lines indicate failed links. In Figure 1, the fact that node 02 is darkened indicates that it has failed. All other nodes are active. The three root links (25, 26 and 27) are all active, but root link 25, connected to Channel A, is currently the active interface to the network and is shown as a thicker line. If a VT240 terminal is available, failed nodes and links are colored red, active links are solid green lines, and idle links are dashed green lines. The network status display can be requested by an operator command to the GPC local operating system via the VT100 or VT240 and a RS232 link. The local operating system then displays the current status of the network as in Figure 1 and continues to update that display as links or nodes fail and reconfiguration takes place.

The display is derived from the Network Manager's view of the status of the network hardware as read from I/O Network Status (Section 3.4.1). The display process periodically queries the I/O Network Status process about changes in the status of the network. If changes have occurred since the last time the display process obtained status information, it obtains a new copy of the status information record. It compares its previous copy of the status record to the new copy to detect which network components have a new value of status and then updates the display accordingly. Thus the display is not completely redrawn each time network status changes which produces a significant gain in the response time of the display.

The network displays are presently specific to a particular network topology. The position of each node and link is known in advance. This is in keeping with the fact that the network topologies are presently also defined by static databases. However, it is possible to enhance the present implementation by the use of more flexible data structures to represent a network topology and by providing algorithms to dynamically deduce the topology of any network connected to a GPC. When these capabilities are in place, a logical next step is to implement a correspondingly flexible network display process.

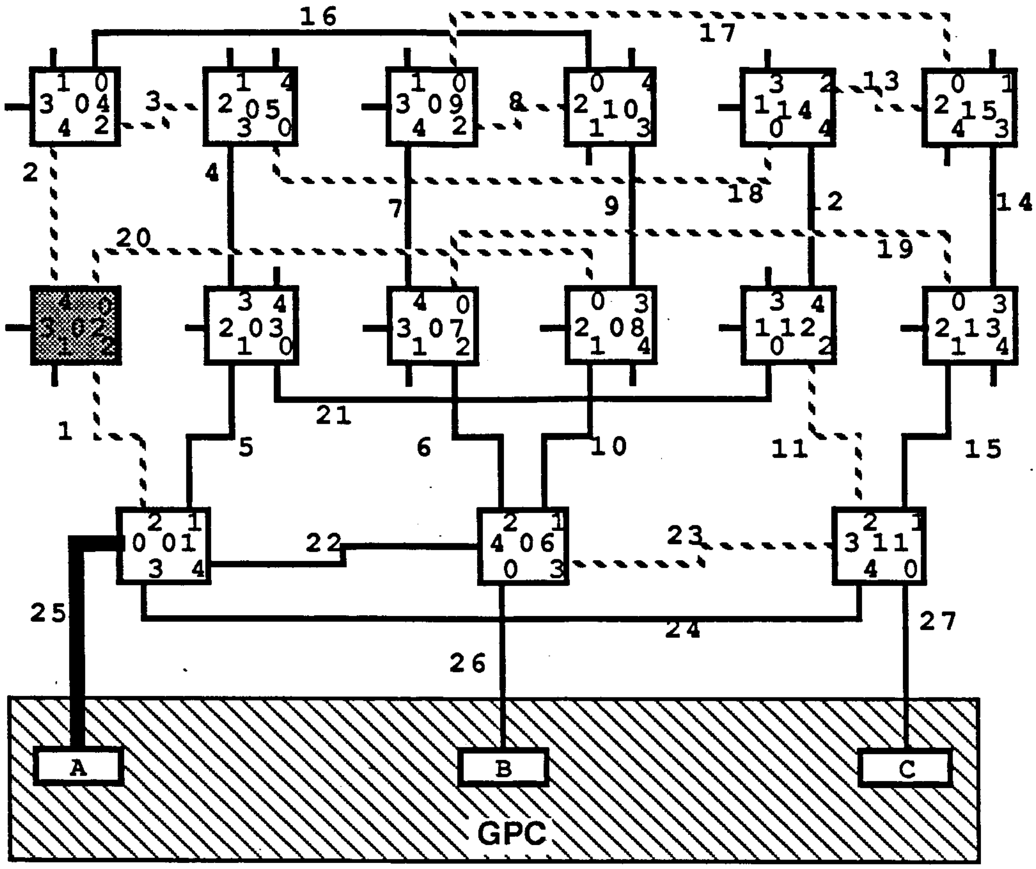


Figure 20. I/O Network Display

3.7 I/O Network Data Dictionary

Active Root Link : A record containing the channel number and the channel identifier of the FTP channel which is currently being used to access a given network via an IOS connected to that channel.

Active Root Link Flag: A Boolean valued flag indicating whether or not a working connection from a GPC to a root node exists.

Available I/O Services : An array of booleans indexed by I/O Service identifiers, one for each GPC in the system. When the boolean is true, the given service is available to the GPC.

Babbler Kcport : A record containing the results of a test which can detect the presence of a babbler on a network. It indicates whether or not the IOS is failed. If it is failed, it indicates whether the failure is due to a failed FTP channel or failed IOS hardware. If the IOS is not failed, it indicates whether or not a babbler was detected on the network by the IOS.

BABBLER_REPORT (INTERFACE_FAILURE)

```
Case INTERFACE_FAILURE
  when TRUE =>
     INTERFACE_FAILURE_TYPE
  when FALSE =>
     BABBLER_DETECTED
```

Branch Record : A discriminated record which contains information about the nodes which lie downline from an outboard of the failed node to which the branch belongs. If any nodes lie on this branch, their node number are entered in a queue in the record. A boolean flag indicates whether or not this branch is reconnected to the network or still requires further network reconfiguration to make these nodes reachable. The number of nodes in this branch is also kept in the record.

BRANCH_RECORD(ANY_NODES_IN_BRANCH))

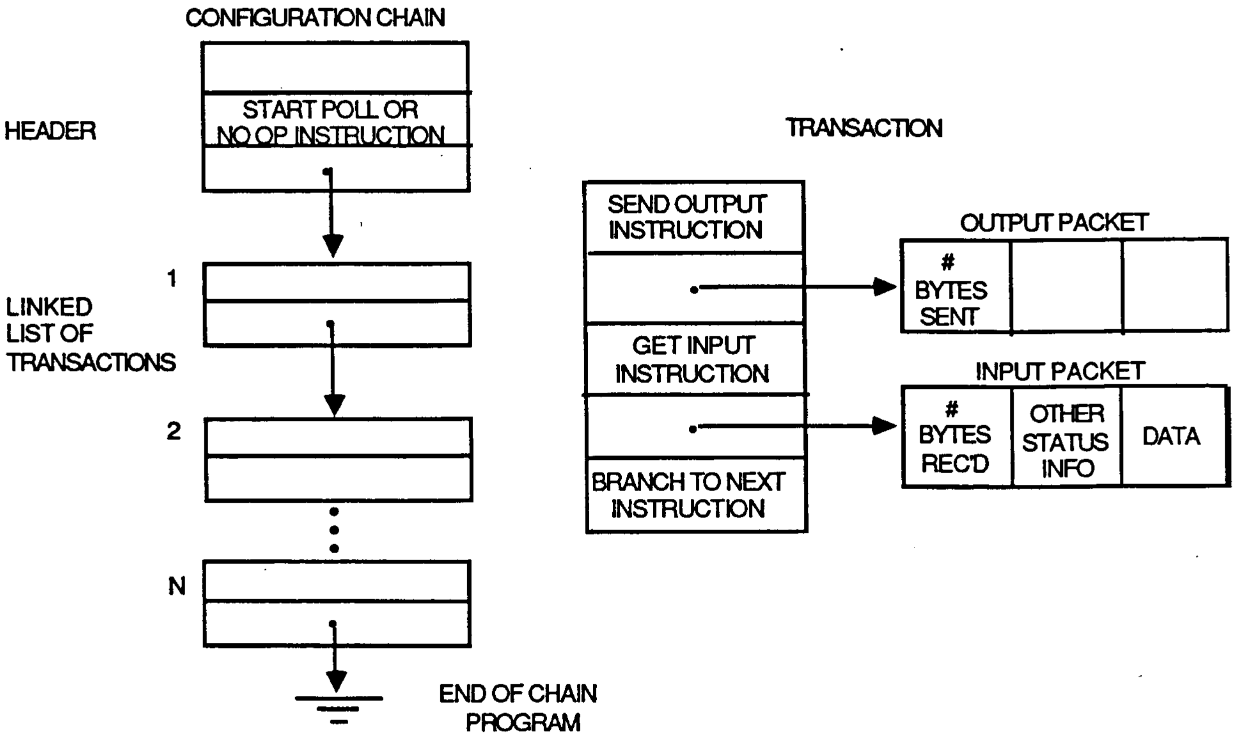
```
Case ANY_NODES_IN_BRANCH
  when FALSE =>
     null;
  when TRUE =>
    RECONNECTED
    QUEUE_OF_NODES_IN_BRANCH
    NEXT_ENTRY_IN_QUEUE
```

Channel Identifier : An identifier which designates a particular physical channel of an IOP.

Channel Number : A logical identifier for a channel of an IOP which contains the IOS connected to a given network.

Channel Selection : An array indicating which FTP channels interface to a particular network.

Configuration Chain : An IOS program which executes a variable number of transactions on the network for the Network Manager. It is designed to allow the chain to be run with or without contention.



Configuration Commands : An ordered set of formatted messages which the Network Manager wishes to send to nodes in its network. The number of messages can vary from a minimum of one message to a maximum equal to the number of nodes in the network. The messages constitute the output packets transmitted by the Configuration Chain.

NODE_COMMAND_ARRAY_RECORD

COUNT NODE_COMMAND_ARRAY

NODE_COMMAND_ARRAY

1	2	3	NODE MESSAGE	• • •	COUNT

Configuration Lifetime : The field of the node reconfiguration command message which specifies how long the node should keep the port configuration also specified in the message in effect. A value of one means that the change in configuration is permanent and a value of zero means the change is in effect only until the response to this command has been transmitted whereupon the previous port configuration is restored.

Configuration Report : A record containing the results of the attempt to send out a set of Configuration Commands for the Network Manager. The first field indicates whether or not the network interface is failed, and if it is failed, the cause of the failure. If the interface is not failed, another field indicates whether or not a babbler was detected on the network.

If no babbler is detected, the last field is an array of Node Response Records, one for each transaction executed by a Configuration Chain.

CONFIGURATION_REPORT (INTERFACE_FAILURE)

```
Case INTERFACE_FAILURE
 when TRUE =>
   ATTRIBUTE_FAILURE_TO_CHANNEL_OR_IOS
 when FALSE =>
   CONFIG_CHAIN_RECORD
```

CONFIG_CHAIN_RECORD (BABBLER_DETECTED)

```
Case BABBLER_DETECTED
 when TRUE =>
   DETECTED_DURING_CONTENTION_OR_TRANSMISSION
 when FALSE =>
   NODE_RESPONSE_ARRAY_RECORD
```

NODE_RESPONSE_ARRAY_RECORD

```
RESPONSE_COUNT
NODE_RESPONSE_ARRAY
```

NODE_RESPONSE_ARRAY

1	2	NODE RESPONSE RECORD	• • •	RESPONSE COUNT

NODE_RESPONSE_RECORD (HAD_ERROR)

```
Case HAD_ERROR
 when TRUE =>
   NULL
 when FALSE =>
   DATA
```

Connected Networks : An array of booleans indexed by network identifier, one for each GPC in the system. When the boolean is true, the given network is physically connected to the GPC.

Connection Indicator : A boolean valued object which when true indicates that a given network is physically connected to a given GPC.

Contention Option : A boolean flag indicating whether or not the Network Manager wants to execute a configuration with contention for the network, in which case the boolean is true, or without contention, in which case it is false.

Current Root Link : A record containing the channel number and the channel identifier of the FTP channel which is currently being used to access a given network via an IOS connected to that channel.

Display of Log Entries : A display on a terminal or monitor screen of the contents of the most recent entries in the log. The number of entries displayed will depend on the type of screen used in the implementation.

DPM Partition : An object which designates which half of the DPM memory is selected, the lower 4K bytes or the upper 4K bytes.

DPM Pointer : A pointer whose value is the address of the first addressable byte of one DPM or a set of DPMs. When used to read from a DPM, the pointer value selects exactly one physical DPM. When used to write to a DPM, the pointer value may select a set of physical DPMs, at most one per channel, each occupying the same memory space within a channel. The pointer imposes an organization on the memory space which supports the execution of chains on an I/O Network and the reading and writing of data used by those chains.

Ending Address of DPM Memory Block : The address of the last byte in a block of contiguous Dual Ported Memory locations.

Error Analysis Report : A discriminated record summarizing the results of the error analysis performed on the data contained in the Status Collection Report. It indicates whether or not any errors were detected and if they were, whether or not the analysis is conclusive. When an analysis is conclusive, the type of fault and other information relating to the source of the error(s) is provided.

ERROR_REPORT_RECORD (STATUS)

```
Case STATUS
  when NO_ERRORS =>
      null
  when ANALYSIS_UNSUCCESSFUL =>
      null
  when ANALYSIS_SUCCESSFUL =>
      FAULT_ANALYSIS_RECORD
```

FAULT_ANALYSIS_RECORD (FAULT)

```
Case FAULT
  when NO_FAULTS =>
      null;
  when BABBLER =>
      null;
  when ROOT_LINK_FAILURE =>
     FAILED_CHANNEL
     ATTRIBUTED_TO
  when LINK_FAILURE =>
     FAILED_ROOT
     FAILED_INBOARD_PORT
     FAILED_NODES
     FAILED_NODE_SET
  when TALKS_OUT_DISABLED_PORT =>
     FAILED_NODE
  when SINGLE_NODE_FAILURE .=>
     FAILED_NODE
  when BAD_ADDR_OR_CONFIG_DATA =>
     FAILED_NODE
```

Fast Grow Option : A Boolean valued variable which when true directs the network to be grown without performing any diagnostic tests.

GPC Identifier :A logical identifier which is uniquely assigned to every GPC in the system.

Inboard Port of Node Under Test : The port on which the node under test receives messages transmitted by the active IOS of the network.

Initialized IOS/DPM for Network Manager : A dual ported memory to which programs and data for executing chains for the I/O Network Manager have been written.

I/O Network Identifier : A logical identifier which is uniquely assigned to every physical network in the system.

I/O Service Descriptor : A record which states whether a given I/O Service is local or regional. In the case of a local I/O network, it contains an array of network identifiers which specify the networks assigned to this service.

IO_SERVICE_DESCRIPTOR (SERVICE)

```
Case SERVICE
 when REGIONAL =>
   NETWORK_ID
 when LOCAL =>
   NETWORK_ID_ARRAY
```

NETWORK_ID_ARRAY

1	2	NETWORK ID TYPE	• • •	NUMBER OF NETWORKS

I/O Service Identifier : A logical identifier which is uniquely assigned to every I/O Service in the system.

IOS Identifier : An logical identifier which designates a particular IOS which in turn maps to a specific address range within an FTP channel.

Link Enabled Record: A discriminated record whose discriminant is a Boolean valued flag. If the value is true, the link was enabled; otherwise a second field indicates the reason why the link is not enabled.

Log Entry : The information passed to the I/O Error Log or the I/O Event Log by various subprograms in the system. It contains the I/O Network Identifier, a time stamp, a field indicating the subprogram which made the entry, a field for comments describing the reason for the entry, and fields for up to four bytes of data which the logging subprogram wishes to record.

Logging Enable : A boolean flag which indicates whether or not errors which are detected should be logged to the I/O Error Log.

Long DPM Address : A record which maps the thirty-two bit address space of the M680X0 microprocessor into specific fields which can be used to generate pointers which will allow the FTP to access a given IOS/DPM.

Maximum Retries : A positive integer which indicates the maximum number of times to try to reconfigure the network.

Network Configuration: A table which holds the current configuration of each port of each node in a network. A port may be enabled, in which case its configuration with respect to the GPC is either inboard or outboard, or disabled, in which case its configuration is idleport. A properly functioning node will receive data transmitted by the GPC on an inboard port and retransmit that data on all its outboard ports. A node may therefore have at most one inboard port but several outboard ports. The inboard/outboard distinction is from the point of view of the Network Manager only; the node hardware does not make this distinction.

NETWORK_CONFIGURATION

1	2	PORT CONFIG ARRAY	●●●	NUMBER OF NODES

PORT_CONFIG_ARRAY

1	2	PORT CONFIG TYPE	●●●	NUMBER OF PORTS

Network Display Database : A database which contains information about the position of nodes, ports and links in the graphics display of the network status.

Network Interface Status : An object which holds the status of all the interfaces to a given network. The status of an interface may be active, idle, available, failed ios, or failed channel.

NETWORK_INTERFACE_STATUS

1	2	NETWORK INTERFACE STATUS RECORD	● ● ●	NUMBER OF CHANNELS

NETWORK_INTERFACE_STATUS_RECORD

CHANNEL IOS STATUS

Network Is Active Flag : A Boolean valued variable indicating whether or not the network has been initialized grown by the I/O Network Manger.

Network Status : A record consisting of Network Interface Status, Node Status, and the channel with the active root link to the network.

Network Subscribers : A list of nodes to which either a DIU or a remote GPC is connected. The port number used in the connection is also given for each subscriber.

Network Topology : A table which describes the physical makeup of an I/O Network. It consists of an array of records, one for each node in the network. The record holds the physical address of the node and a port by port description of the network element adjacent to the node through the port. The information in this table should be sufficient to grow and maintain the network, to deduce the rootlinks to the network from a given GPC, and to deduce the set of DIUs which are reachable through this network. The data in this table is static; it does not change over time. Therefore, to save memory space and unnecessary copying of data, it should be referenced by means of pointers.

NETWORK TOPOLOGY

1	2	NODE RECORD	• • •	NUMBER of NODES

NODE RECORD

	NODE ADDRESS				
PORT ARRAY	1	2	PORT RECORD	• • •	NUMBER OF PORTS

PORT RECORD (ADJACENT_ELEMENT)

```
Case ADJACENT_ELEMENT
  when NODE =>
    NODE_NUMBER
    NODE_ADDRESS
    PORT_NUMBER
  when GPC =>
    GPC_ADDR
    CHANNEL
    IOS
  when DIU =>
    DIU_ADDR
    DIU_ID
```

Network Usability : An object whose value indicates which software process, the I/O Network Manager or the I/O Communication Manager, is allowed to transmit on an I/O network. A network which is in-service or repaired may only be accessed by the I/O Communication Manager. A network which is out-of-service may only be accessed by the Network Manager. A network which is repaired has been acted upon by the Network Manager in response to a call by the I/O Communication Manager.

Node Number : A logical identifier of a node a network. Its scope is for a given network only.

Node Status : An array indicating the status of each node in a given network. A node may be active, idle, or failed. An active node has ports enabled so as to make it part of the virtual bus which carries data on the network. A failed node is a node which is completely non-functional. It is isolated from the rest of the network by disabling ports on adjacent nodes. An idle node is a node whose status is unknown because no attempt has yet been made to reach the node by the network manager.

NODE_STATUS

1	2	NODE STATUS RECORD	• • •	NUMBER OF NODES

NODE_STATUS_RECORD

ADDRESS STATUS PORT_STATUS_ARRAY

PORT_STATUS_ARRAY

1	2	STATUS TYPE	• • •	NUMBER OF PORTS

Node Response Record : A record in which the first field indicates whether or not communication errors were detected when the transaction to the node was executed. If errors were not detected, a second field contains the data which the node returned in response to the command it received.

Node Under Test : The node number of a node to be tested.

Passed Diagnostic Tests : A boolean valued flag which when true indicates that the full set of diagnostic tests have been passed. A value of false indicates that at least one test failed.

Relative DPM address : A record which is used to map the thirty-two bit address used by the FTP to access a location in an IOS/DPM into a sixteen bit value which the IOS will use to access the same location. Since the address space of the IOS is 8K bytes, only the lower thirteen bits are used in the mapping, the three highest order bits are assigned a value of zero. The mapping is defined below, where f is the value of the i^{th} bit in the sixteen bit address:

$$f(i)= \begin{array}{l} 0, \text{ if } 15 <= i <= 13 \\ \text{value of the } i^{th} \text{ bit in the thirty-two bit address if } 0 <= i <= 11 \\ \text{value of the } 15^{th} \text{ bit in the thirty-two bit address if } i= 12 \end{array}$$

Restore Record: A record containing information about the repaired network component which the operator wishes to be returned to service. If a node is to be restored, the node number is provided. If a link is to be restored, a node number and a port number adjacent to that link is provided.

RESTORE_RECORD (NODE_OR_LINK)

```
Case NODE_OR_LINK
  when NODE =>
    Node Number
  when LINK =>
    Node Number
    Port Number
```

Results of Channel OK Test : A boolean value which is true if the test indicates that an FTP channel is not desynchronized and false if it is desynchronized.

Results of DPM Block Test : A boolean value which is true is the test is passed and false otherwise.

Results of DPM Memory Tests : A boolean value which is true is the test is passed and false otherwise.

Results of DPM Word Test : A boolean value which is true is the test is passed and false otherwise.

Results of Spare Link Chain : The data and status returned by the I/O Communications Manager after the Spare Link Chain has been executed on the network.

Root Link History : A table maintained by the Network Manager with an entry for each root link in its network. The entry holds a tally of errors attributable to the IOS connected to that root link.

Root Links : An array of records one for each rootlink from a GPC to an I/O network. Each record contains information about the physical makeup of that rootlink. It contains the node address of the root node, the port of the root node used to connect to the IOS, the IOS identifier, and the physical channel containing the root link.

ROOT_LINKS

1	2	ROOT LINK RECORD	• • •	NUMBER OF CHANNELS

ROOT_LINK_RECORD

```
CHANNEL
IOS
NODE_NUMBER
NODE_ADDRESS
PORT_NUMBER
```

Spare Link Cycling Log : A log with an entry for each link in the network. The value of an entry can be either cycled or not cycled. A value of cycled means that the link has spent at least one active cycling period in the network in the current cycle or that the link is failed. A value of not cycled means that the link is ready to be activated.

Spawning Node : A node from which further network growth is taking place or a node whose port is activated first when a link is being enabled.

Spawning Port : The port which is enabled first when a link is enabled. This port in turn retransmits the reconfiguration command to the adjacent (target) node which enables the adjacent port.

Spawning Queue : A queue into which node identifiers are placed in the order in which those nodes are to be added to the network.

Starting Address of DPM Memory Block : The address of the first byte in a block of contiguous Dual Ported Memory locations.

Status Collection Report : A record containing the results of the attempt to send out a set of Status Collection Commands for the Network Manager. The first field indicates whether or not the network interface is failed, and if it is failed, the cause of the failure. If the interface is not failed, another field indicates whether or not a babbler was detected on the network. If no babbler is detected, the last field is an array of Node Response Records, one for each selected status transaction executed by a Status Collection Chain.

STATUS_COLLECTION_REPORT (INTERFACE_FAILURE)

```
Case INTERFACE_FAILURE
 when TRUE =>
   ATTRIBUTED_FAILURE_TO_CHANNEL_OR_IOS
 when FALSE =>
   STATUS_CHAIN_RECORD
```

STATUS_CHAIN_RECORD (BABBLER_DETECTED)

```
Case BABBLER_DETECTED
 when TRUE =>
   DETECTED_DURING_CONTENTION_OR_TRANSMISSION
 when FALSE =>
   NODE_STATUS_ARRAY_RECORD
```

NODE_STATUS_ARRAY_RECORD

```
COUNT
NODE_STATUS_ARRAY(1..COUNT)
```

NODE_STATUS_ARRAY

1	2	NODE STATUS RECORD	• • •	COUNT

System Address : A thirty-two bit value which maps to some physical location in the system. By which the M680X0 microprocessor accesses those physical locations.

Target Node : A node being added to the network during network growth or a node whose port is activated last when a link is being enabled.

Target Port : The port which is enabled last when a link is enabled. This port receives the reconfiguration command from the adjacent (spawning) node which causes it to be enabled.

Unreachable DIUs : A list of DIUs which are attached to failed nodes and which therefore cannot send or receive messages on the I/O network.

Update Generation : A value associated with each call made by an I/O Network Manager to update the Network Status in the I/O Network Status process. Each time the status is updated, a unique generation number is assigned to the current version of the status. Other processes which wish to detect a change in the value of the Network Status may keep a copy of the generation of the last status value they have obtained from I/O Network Status. If the generation of this copy of network status is not equal to the generation of the current copy of status maintained by I/O Network Status, then a new value of status could be obtained by the process.

Updated Status Chain : A status collection chain which has been modified to deselect specific node status transactions.

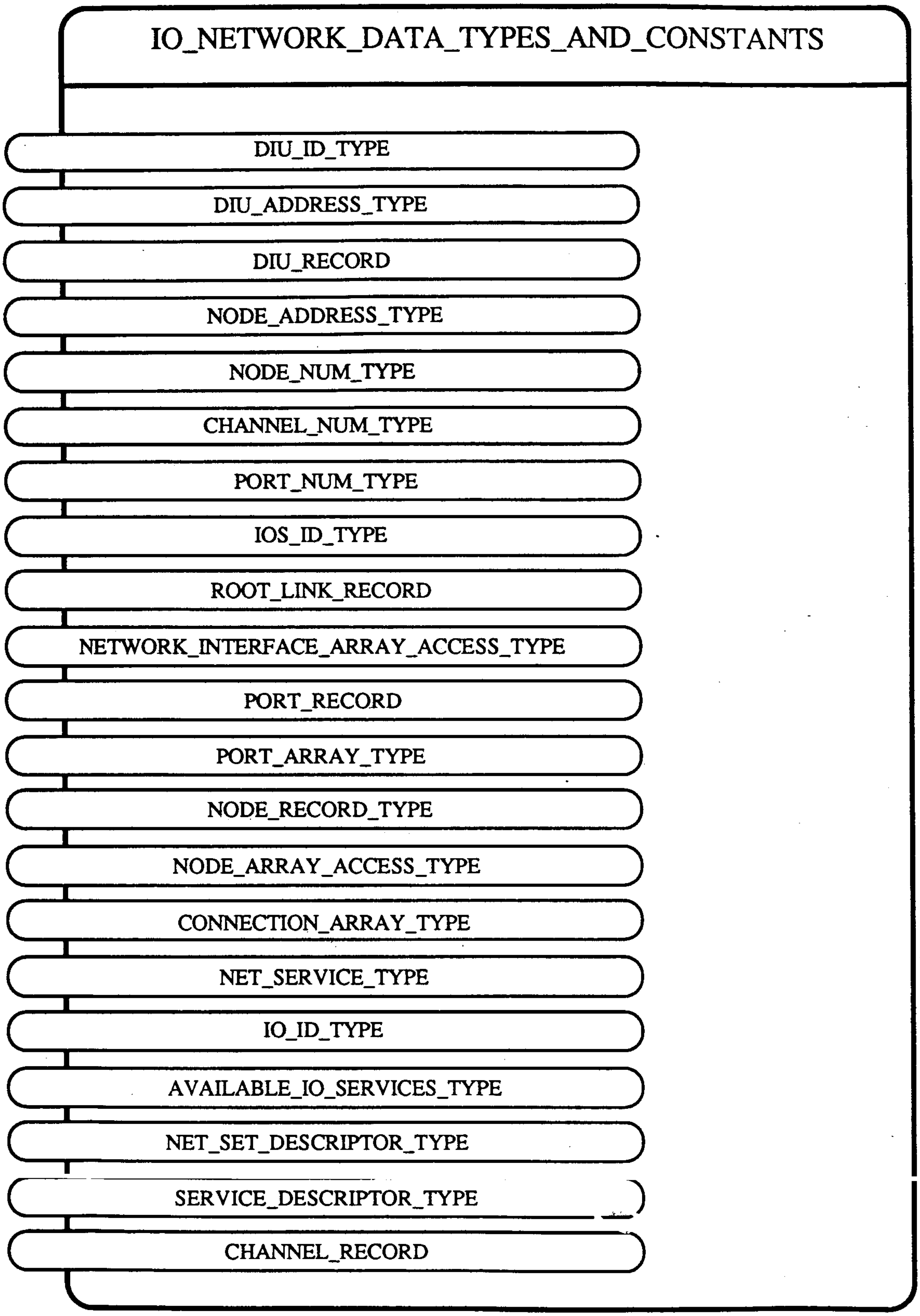
Updated Status Flag : A Boolean valued flag which indicates whether or not the Network Status of a particular network has been updated by the Network Manager since the last time this status was obtained.

4.0 ADA IMPLEMENTATION OF THE I/O NETWORK MANAGER

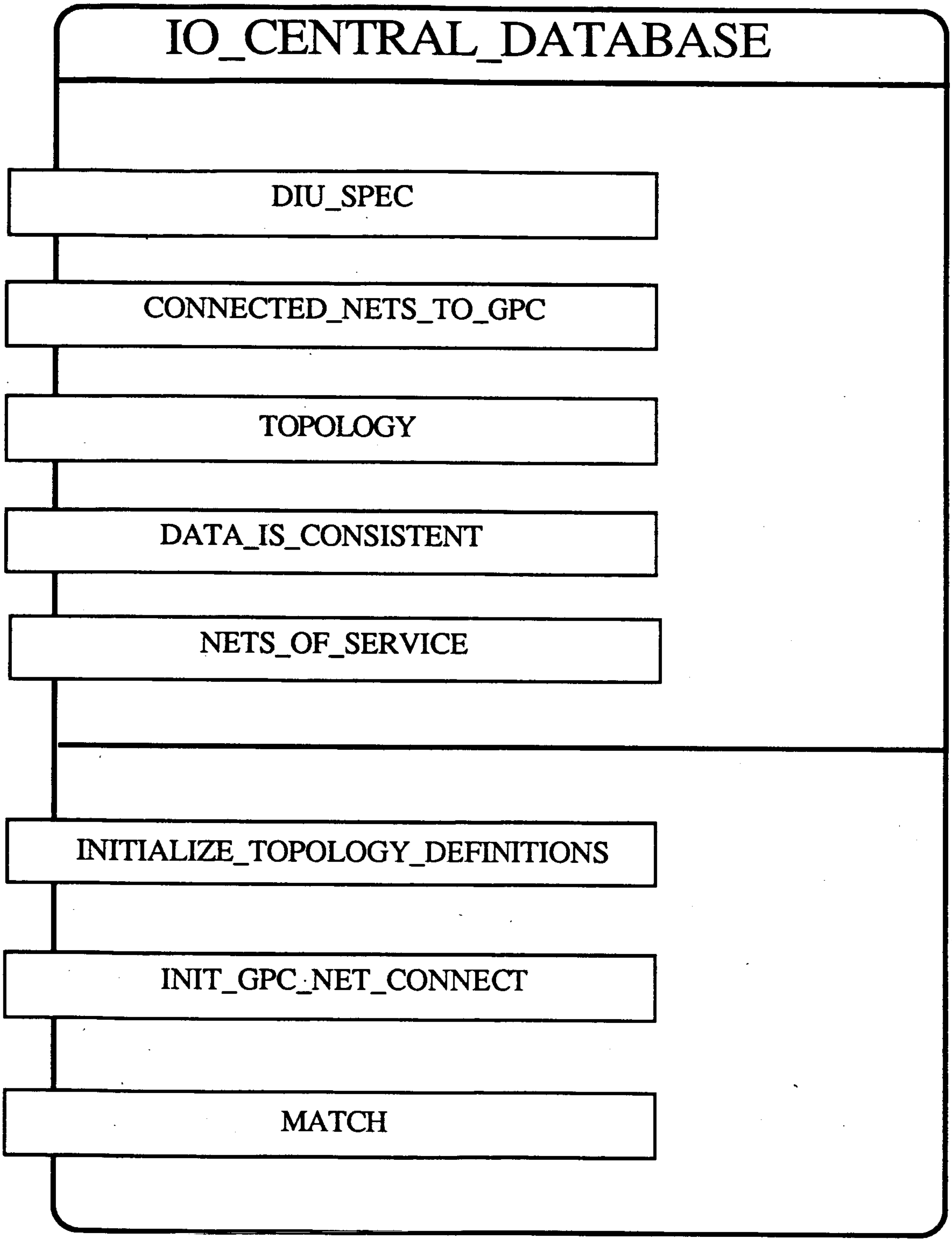
This section bridges the gap between the software specifications of Chapter 3 and the Ada® code used to implement those specifications. The Ada® language is widely advertised as a design language.[1] Grady Booch has put forward a high level diagramatic design methodology. This methodology has been used to map the I/O Network management software specifications into Ada packages, tasks and subprograms. This mapping is represented by the Booch diagrams contained in this section.

These high level Ada® constructs are the framework supporting the detailed Ada implementation, i.e. the code which embodies the software specification. The logic described in the software specification is further realized by the Nassi/Shneiderman diagrams, also contained in this section, which convert this logic into standard programming control structures. The logic of the Nassi/Shneiderman diagrams could be coded in any language, while still retaining the high level Ada® organization of the overall design.

[1] Booch, p. 46



Software Specification Reference Number: 3.7



Software Specification Reference Number: 3.3.1

Subprogram: NETS_OF_SERVICE
Inputs: service identifier
Outputs: array of network identifier

Returns an array of network ids associatied with the specified service id

Subprogram: TOPOLOGY
Inputs: network identifier
Outputs: a pointer to a network topology

Returns a pointer to the specified network's node array

Subprogram: DIU_SPEC
Inputs: diu identifier
Outputs: diu address
network identifier

Return the address of the diu and the network that the
DIU is connected to for the passed in diu identifier

Software Specification Reference Number: 3.3.1

Subprogram: DATA_IS_CONSISTENT
Inputs: network identifier
Outputs: boolean flag

```
for <each node in the net>
    for <each port in the node>
        ?? Does the global database indicate that this port is
           connected to another node ??
        yes
            Call MATCH to see if the data from the port agrees with the database.
        no
            ?? Does the global database indicate that this
               port is connected to a GPC ??
            yes
                Set GPC_PRESENT flag.
            no
?? Has GPC_PRESENT been set ??
yes
    Return 'true' to signal that data is consistent.
no
    Raise NO_GPC exception
```

DATA_IS_CONSISTENT exception handler

```
?? Is the exception a NO_GPC exception ??
yes
    Log "No GPC" error
no
    Log generic error message
```

Subprogram: CONNECTED_NETS_TO_GPC
Inputs: gpc identifier
Outputs: array of booleans

```
Returns an array of booleans where a TRUE entry means
the net is connected to the specified GPC.
```

Software Specification Reference Number: 3.3.1

Subprogram: INIT_GPC_NET_CONNECT
Inputs: none
Outputs: none

```
Initialize GPC_NET_CONNECT array to all elements "false"
for <range of network id types>
    for <the number of nodes in each network type>
        for <the number of ports in each node>
            ?? Is the port which corresponds to these network,
            node, and port numbers connected to a GPC ??
            yes
                Set the element of
                GPC_NET_CONNECT which
                corresponds to the port
                being considered to 'true'..
            no
```

Subprogram: INITIALIZE_TOPOLOGY_DEFINITIONS
Inputs: none
Outputs: none

```
for < each network in the system >
    for < each node in the network >
        for < each port in the node >
            Assign values to port records specifying
            parameters of adjacent element.
```

Software Specification Reference Number: 3.3.1

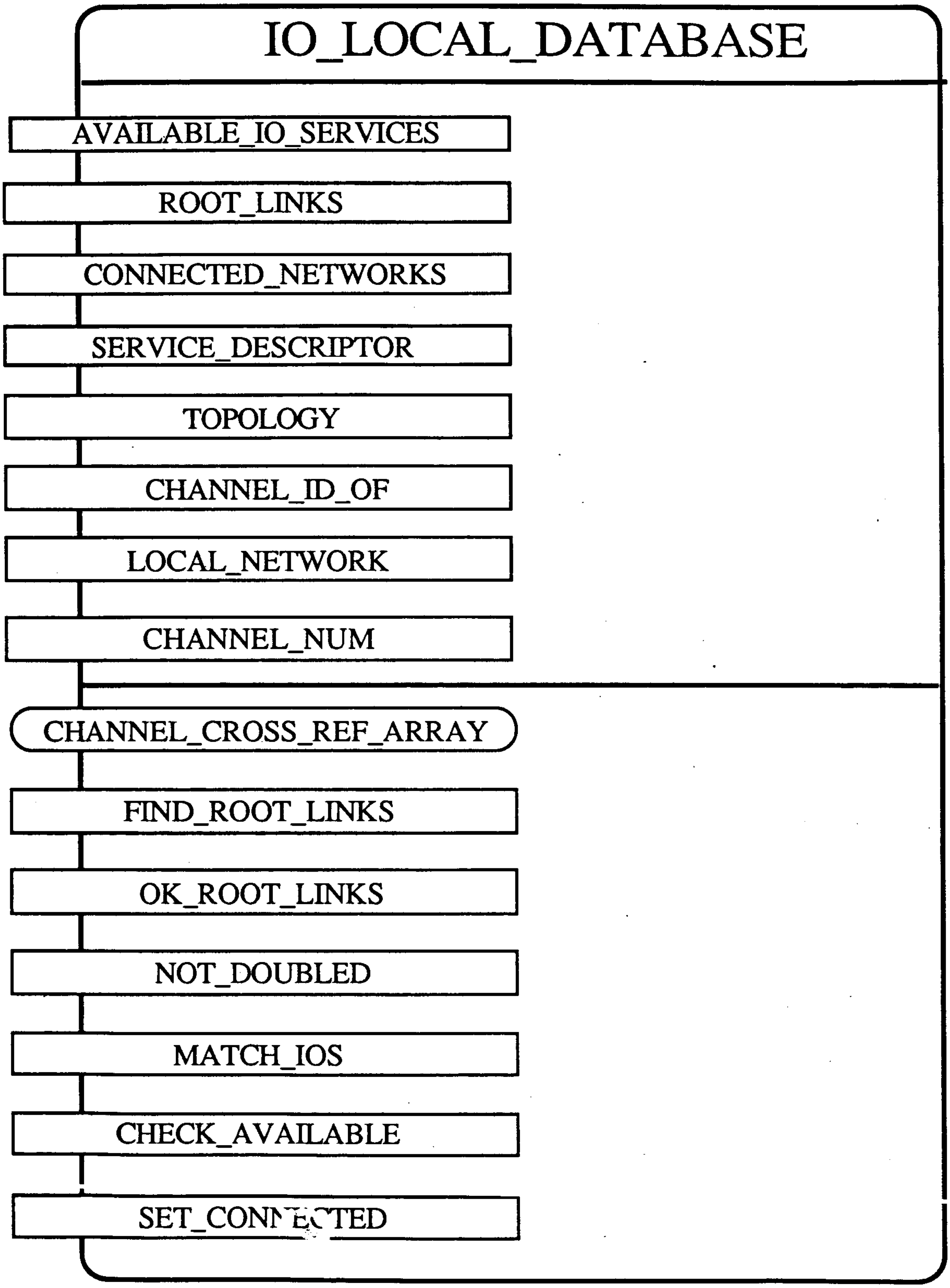
Subprogram: MATCH
Inputs: node
port
Outputs: none

```
?? Does the node say that it is connected to a node whose number is in range ??
    no
        raise
        NODE_NUM_ERROR
    yes
        ?? Do the two nodes believe that they are connected to a node
            no
                MATCH_CHECK = false
                raise
                ELEMENT_ERROR
            yes
                ?? Do the two nodes agree on which ports are connected ??
                    no
                        MATCH_CHECK = false
                        raise
                        NODE_ERROR
                    yes
                        ?? Do the two nodes agree that they are connected to each other ??
                            no
                                MATCH_CHECK = false
                                raise
                                PORT_ERROR
                            yes
                                ?? Do the two nodes agree on the logical address of the second node ??
                                    no
                                        MATCH_CHECK = false
                                        raise ADDR_ERROR
                                    yes
```

Exception handler for MATCH

?? What type of exception occurred ??				
ELEMENT_ERROR	NODE_ERROR	PORT_ERROR	ADDR_ERROR	NODE_NUM_ERROR
Log element error	Log node error	Log port error	Log address error	Log node number error

Software Specification Reference Number: 3.3.1



Software Specification Reference Number: 3.3.2

Subprogram: AVAILABLE_IO_SERVICES
Inputs: none
Outputs: i/o services available on this GPC

Returns a structure which marks all existing io services as available or not_available on the local gpc

Subprogram: ROOT_LINKS
Inputs: network identifier
Outputs: root links

Provides the root links of the network whose id is passed as a parameter

Subprogram: CONNECTED_NETWORKS
Inputs: none
Outputs: array of booleans

Returns an array of booleans indicating which networks are connected to this GPC

Subprogram: SERVICE_DESCRIPTOR
Inputs: io identifier
Outputs: a service descriptor

Accepts an index into the io service table and returns a service descriptor which contains one or more network id's associated with this io service

Subprogram: TOPOLOGY
Inputs: network identifier
Outputs: a pointer to network topology

?? Is the network identifier conncted to the network ??	
no	yes
return null	return a pointer to the network node array

Software Specification Reference Number: 3.3.2

Subprogram: CHANNEL_ID_OF
Inputs: network identifier
channel number
Outputs: channel identifier

Returns the channel identifier of given channel number and network

Subprogram: LOCAL_NETWORK
Inputs: network identifier
Outputs: boolean flag

Returns a TRUE if the network is local to the GPC
and FALSE if it is regional or not connected

Subprogram: CHANNEL_NUM
Inputs: channel identifier
network identifier
Outputs: channel number

Returns the entry from the channel cross reference array associated
with the given channel identifier and network identifier

Software Specification Reference Number: 3.3.2

Subprogram: FIND_ROOT_LINKS
Inputs: network identifier
Outputs: none

```
for <the number of nodes in the net>
    for <the number of ports per node>
        ?? Is the port connected to a GPC ??
        no
        yes
            ?? Is it my GPC ??
            no
            yes
                Assign temporary variables to store location of root link.
Create local table for the root links, using just enough space to hold them.
Transfer the information about the root.links into the table.
?? Are there any root links in the table ??
no
yes
    ?? Are the root links legitimate ??
    no
        raise ROOT_LINKS_NOT_OK
    yes
        Assign channel numbers to channel cross reference array
```

Subprogram: OK_ROOT_LINKS
Inputs: none
Outputs: boolean flag

```
Return the logical and of the results of applying the  functions  MATCH_IOS and NOT_DOUBLED to the net
```

Subprogram: NOT_DOUBLED
Inputs: none
Outputs: boolean flag

```
For  CHAN1 = <the number of channels in the net>
    For CHAN2 = <CHAN1+1> to <the last channel in the net>
        ?? Is the same root link attached to both channels ??
        yes
            report error
            raise
            TWO_ROOTS_TO_SAME_CHANNEL
        no
            ?? Do the root links from both channels
            connect to the same node ??
            yes
                report error
                raise TWO_ROOTS_TO_SAME_NODE
            no
return true
```

Exception handler for OK_ROOT_LINKS

```
?? Is the exception an IOS_DONT_MATCH error ??
yes
    report error
    return false
no
    return false
```

Subprogram: MATCH_IOS
Inputs none
Outputs: boolean flag

```
for CHAN1 = <the range of channels attached to the net>
    for CHAN2 = <CHAN1+1> to <the last channel in the net>
        ?? Do both channels have the same IOS ??
        no
        yes
            raise IOS_DONT_MATCH
```

Software Specification Reference Number: 3.3.2

Subprogram: CHECK_AVAILABLE
Inputs: io identifier
Outputs: boolean flag

```
Get a list of the available I/O resources from the central database
?? Are the available resources regional or local ??
    regional
        Use table lookup to see if the
        net is reachable from this GPC
    local
        Call SET_CONNECTED to determmine if the
        net is fully connected.
Return a boolean value, which corresponds to whether or not the I/O nets are properly connected.
```

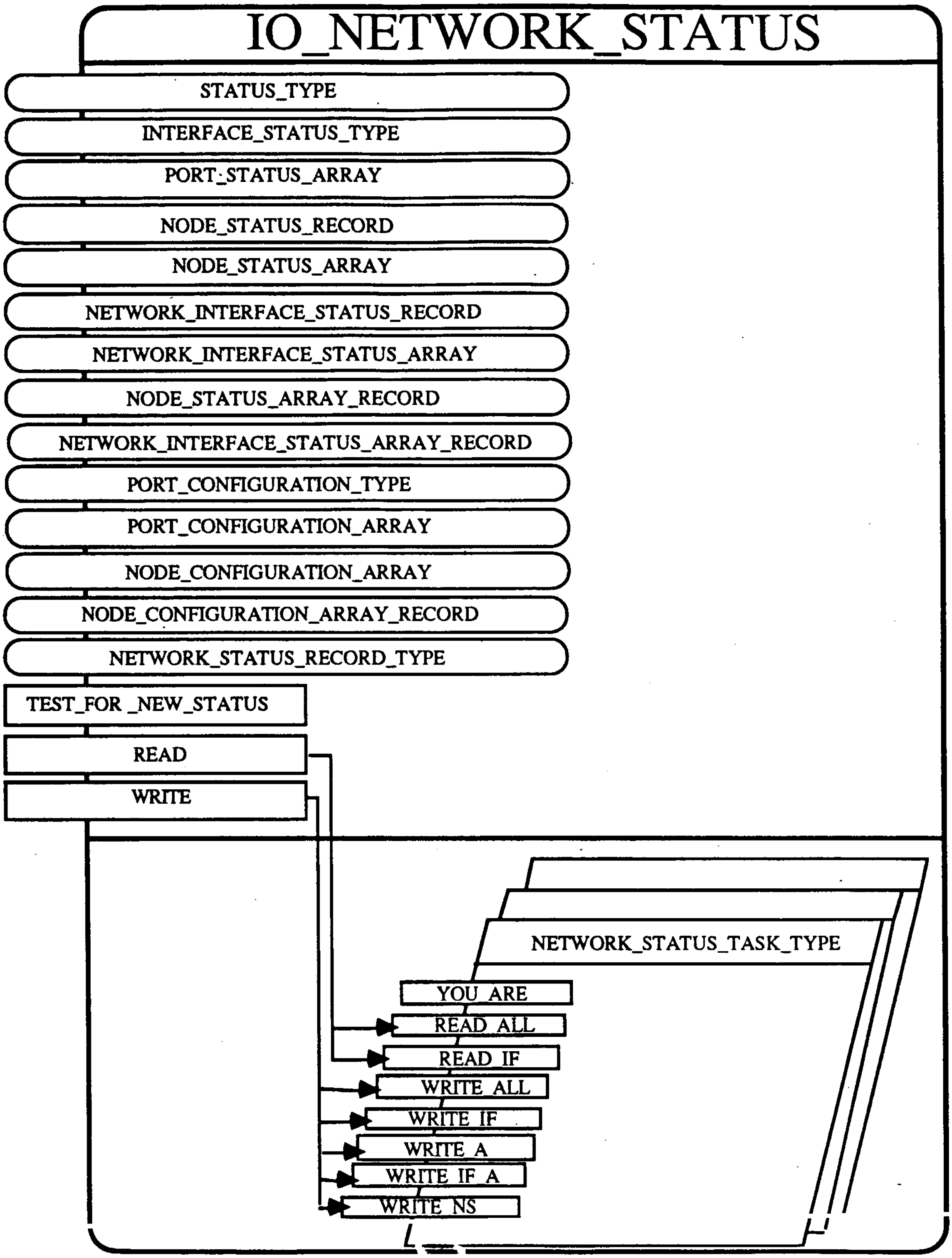
CHECK_AVAILABLE exception handler

```
?? Is the error a MIXED_CONNECTION error ??
    yes
        Report an improper connection error.
    no
        Report that some other type of error occurred.
```

Subprogram: SET_CONNECTED
Inputs: net_set
Outputs: boolean flag

```
For <The number of nets in the service>
    Use table lookup to see if each net is reachsble from this GPC
?? Are all of the  nets reachable from this GPC ??
    yes
        return true
    no
        For <all of the nets in the service>
            Use table lookup to see if each net is reachable from this GPC
        ?? Are any of the nets reachable from this GPC ??
            yes
                raise MIXED_CONNECTION
            no
                return false
```

Software Specification Reference Number: 3.3.2



Software Specification Reference Number: 3.4.1

Subprogram: NETWORK_STATUS_TASK_TYPE

Inputs: network status
interface status
node status

Outputs: active status
protected status
network status

Accept you_are

Assign the network identifier to my_net_id

Accept write_all

Assign the status to protected status

Increment the last written flag for the correspding network idenifier

case < the type of read or write needed >

read if	write all	write if a	write_ns	write a	write_if	read all
Accept read interface status Assign interface status field of protected status to interface status	Accept write all Assign status to protected status Increment last written flag	Accept write interface status with active flag Assign status to interface status field of protected status Assign active to active Increment last written flag	Accept write node status Assign status to node status field of protected status Increment last written flag	Accept write active flag Assign active to active Increment last written flag	Accept write interface status Assign status to protected status Increment last written flag	Accept read all Assign protected status to status Update last updated flag

Software Specification Reference Number: 3.4.1

Subprogram: TEST_FOR_NEW_STATUS

Inputs: i/o network identifier
Outputs: last updated flag
has new value

?? Is last updated flag not equal to last written flag for this network ??	
no	yes
Set has new value to FALSE	Set has new value to TRUE

Subprogram: READ_ALL
Inputs: network identifier
last updated flag
last updated flag
Outputs: network status

?? Is the network connected to this GPC ??	
no	yes
raise network_not_connected_to_this_gpc	Update the network status

Subprogram: READ_IF
Inputs: network identifier
Outputs: interface status

?? Is the network connected to this GPC ??	
no	yes
raise network_not_connected_to_this_gpc	date the interface status

Software Specification Reference Number: 3.4.1

Subprogram: WRITE_NS
Inputs: network identifier
network status
Outputs: none

Update the node status of the network

Subprogram: WRITE_IF
Inputs: network identifier
interface status
Outputs: none

Update the interface and activity status of the network

Subprogram: WRITE_IF_A
Inputs: network identifier
interface status
active
Outputs: none

Update the status of the network

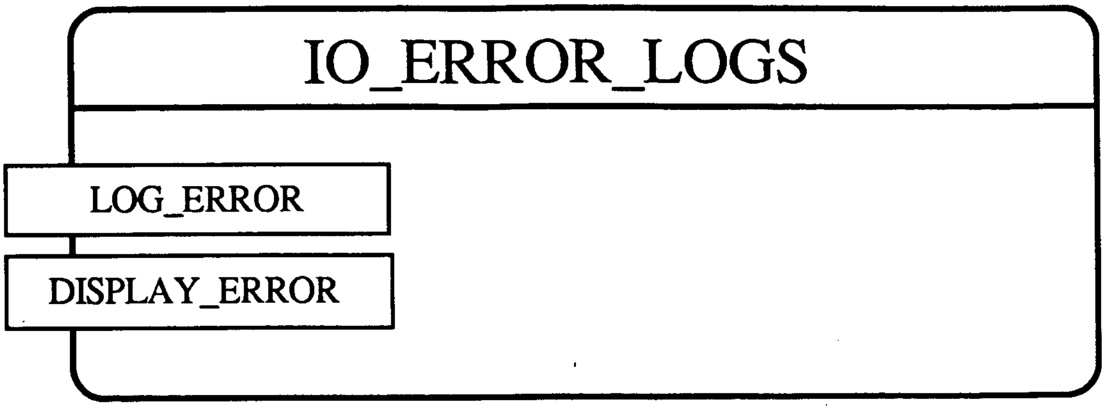
Subprogram: WRITE_ALL
Inputs: network identifier
node status
Outputs: none

Update the status of the network

Subprogram: WRITE_A
Inputs: network identifier
active
Outputs: none

Update the activity status of the network

Software Specification Reference Number: 3.4.1



LOG_ERROR

```
Call LOG_ERROR_ENTRY to log the error.  Overloaded versions of
LOG_ERROR exist to allow different calling conventions.
```

LOG_ERROR_ENTRY

```
?? Is the log index equal to the maximum number of errors that can be stored ??
    yes
        Reset the index to 1.
    no
        Add 1 to the index.
Perform variable assigments to store error information in the error log array.
Add one to the count of total errors, rolling over to zero if an overflow occurs.
```

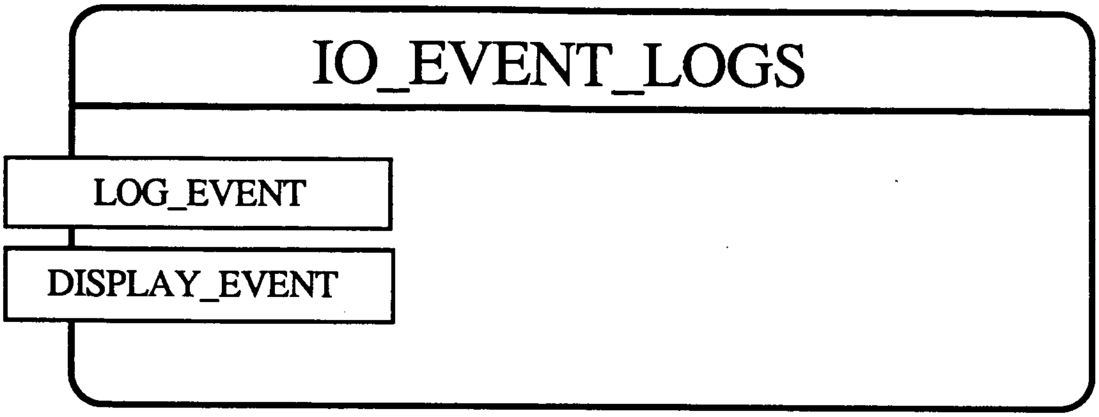
Exception handler for LOG_ERROR_ENTRY

```
Call LOG_EXCEPTION to record the exception.
```

DISPLAY_ERROR_LOG

```
Perform screen initilizations.
Print "IOSS ERROR LOG" as a heading. (double height line)
?? Are there any errors to print ??
    no
    yes
        Print column headings.
        for <the number of errors>
            Print a description of each error, formatting to fit the column headings.
Print a closing line, which reports the total number of errors logged.
```

Software Specification Reference Number: 3.5.1



LOG_EVENT

```
Call LOG_EVENT_ENTRY to log the event Overloaded versions of
LOG_EVENT exist to allow different calling conventions.
```

LOG_EVENT_ENTRY

```
?? Is the log index equal to the maximum number of enents that can be stored ??
    yes
        Reset the index to 1.
    no
        Add 1 to the index.
Perform variable assigments to store event information in the event log array.
Add one to the count of total events, rolling over to zero if an overflow occurs.
```

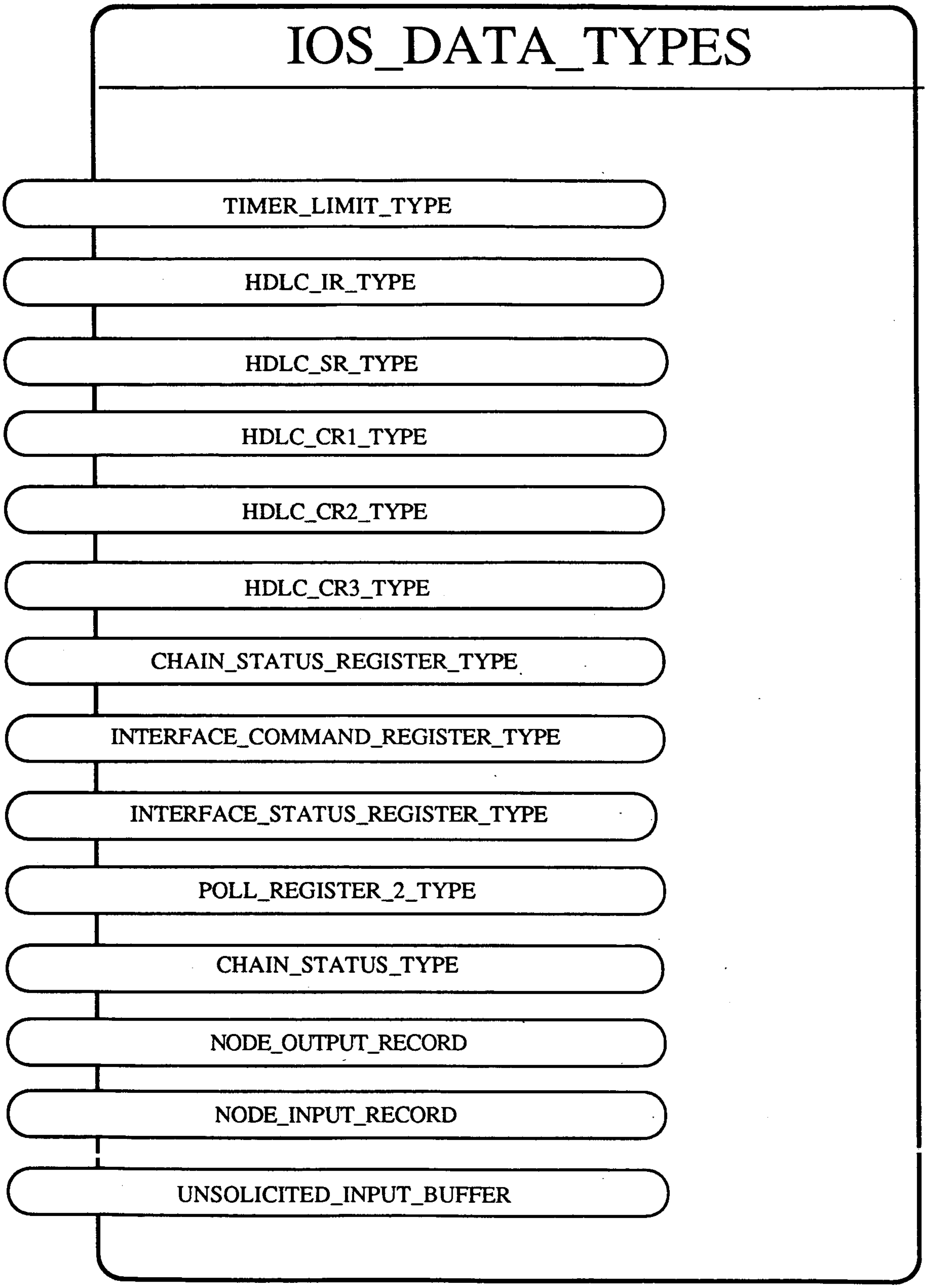
Exception handler for LOG_EVENT_ENTRY

```
Call LOG_EXCEPTION to record the exception.
```

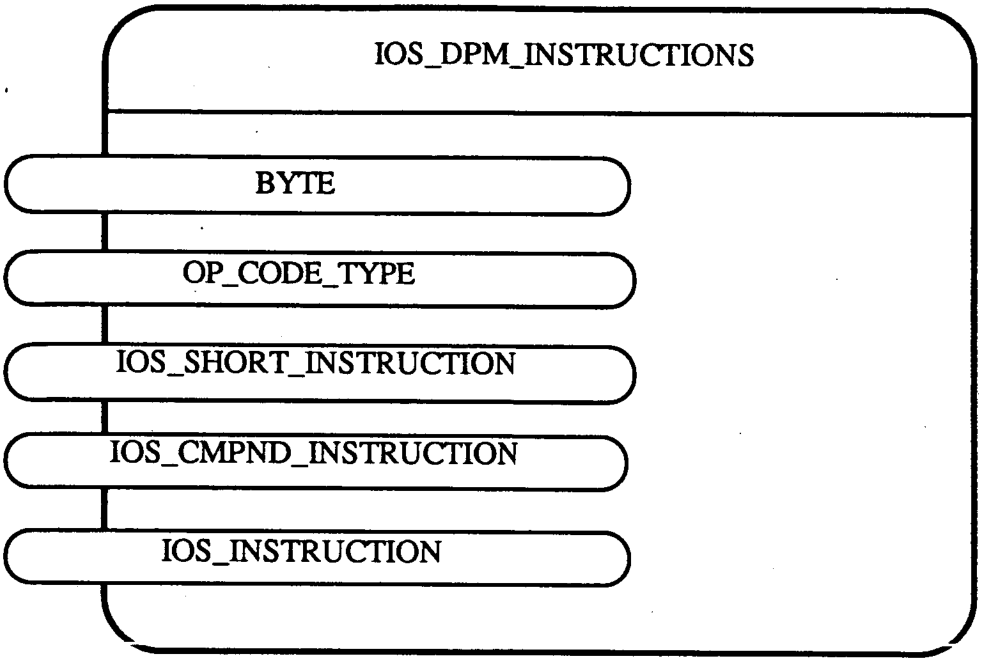
DISPLAY_EVENT_LOG

```
Perform screen initilizations.
Print "IOSS EVENT LOG" as a heading. (double height line)
?? Are there any events to print ??
    no
    yes
        Print column headings.
        for <the number of events>
            Print a description of each event formatting to fit the column headings.
Print a closing line, which reports the total number of events logged.
```

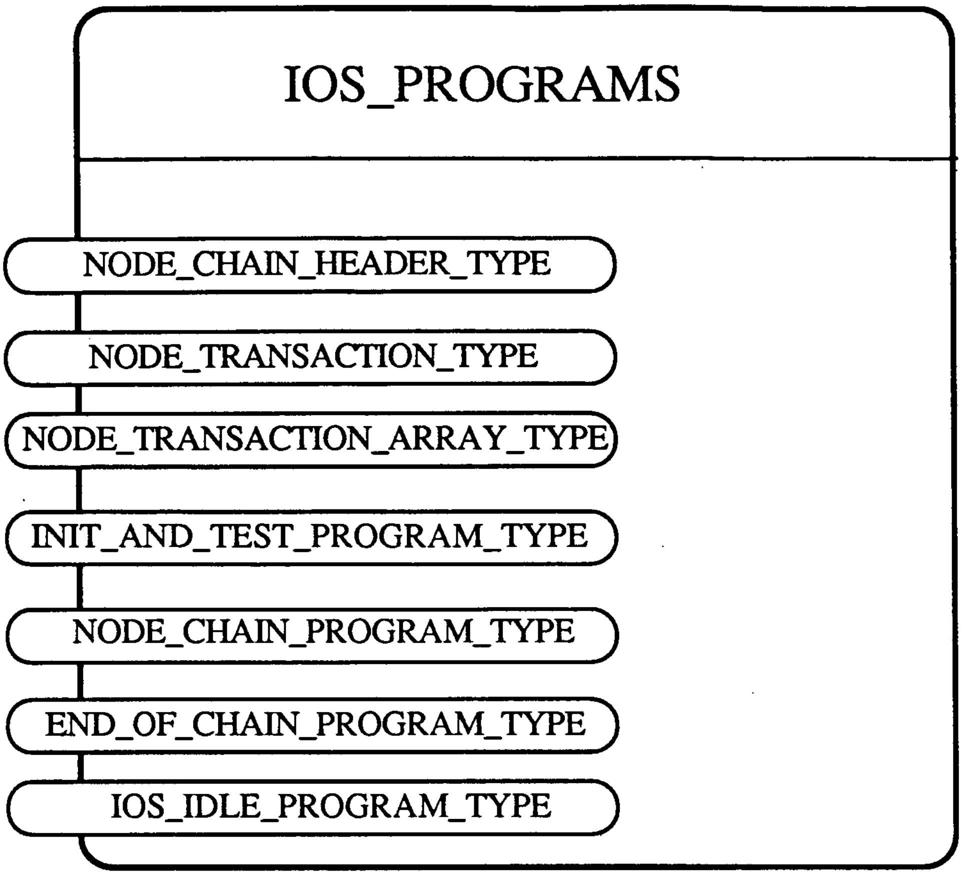
Software Specification Reference Number: 3.5.2



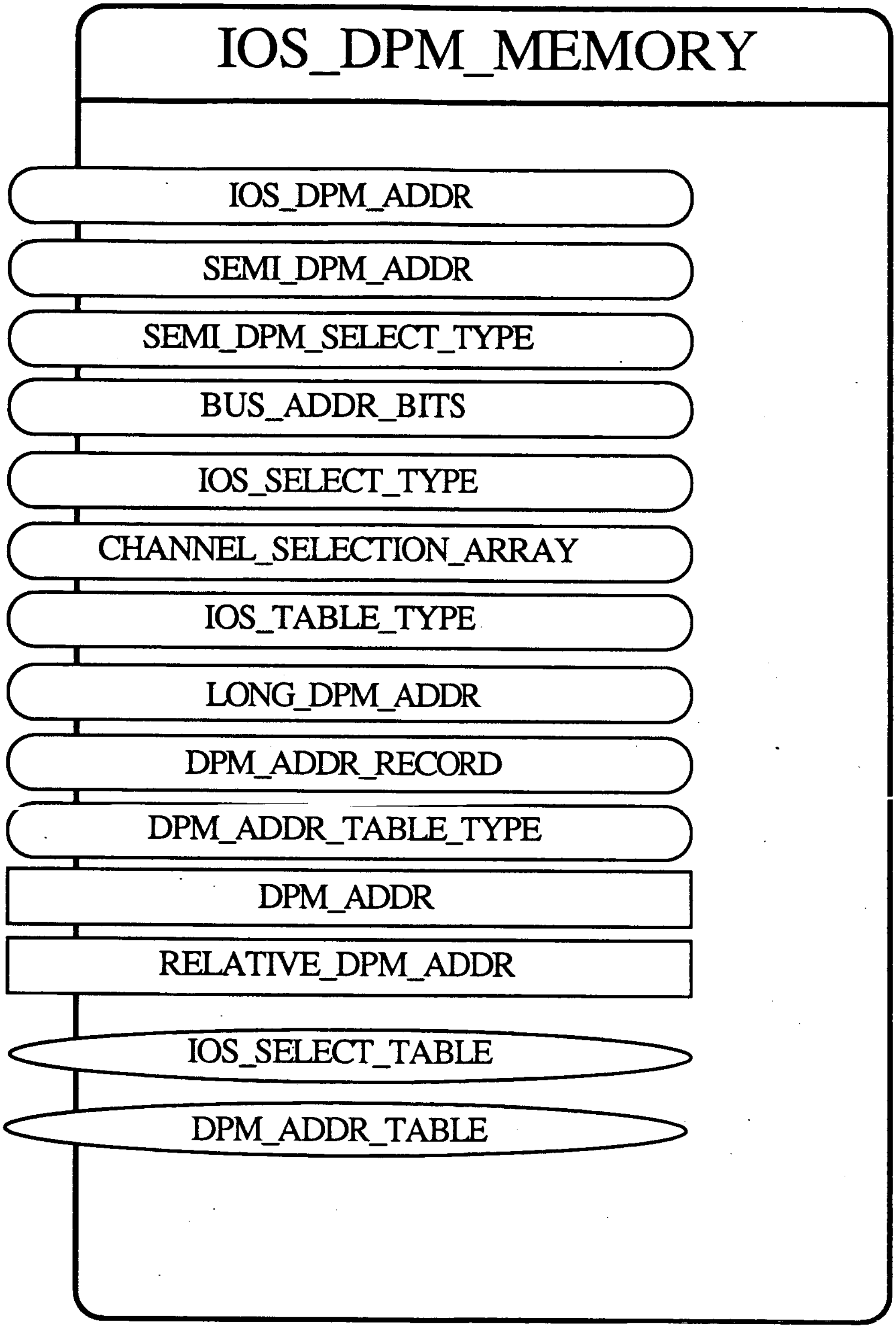
Software Specification Reference Number: 3.2.1.1



Software Specification Reference Number: 3.2.1.2



Software Specification Reference Number: 3.2.1.2



Software Specification Reference Number: 3.2.1.3

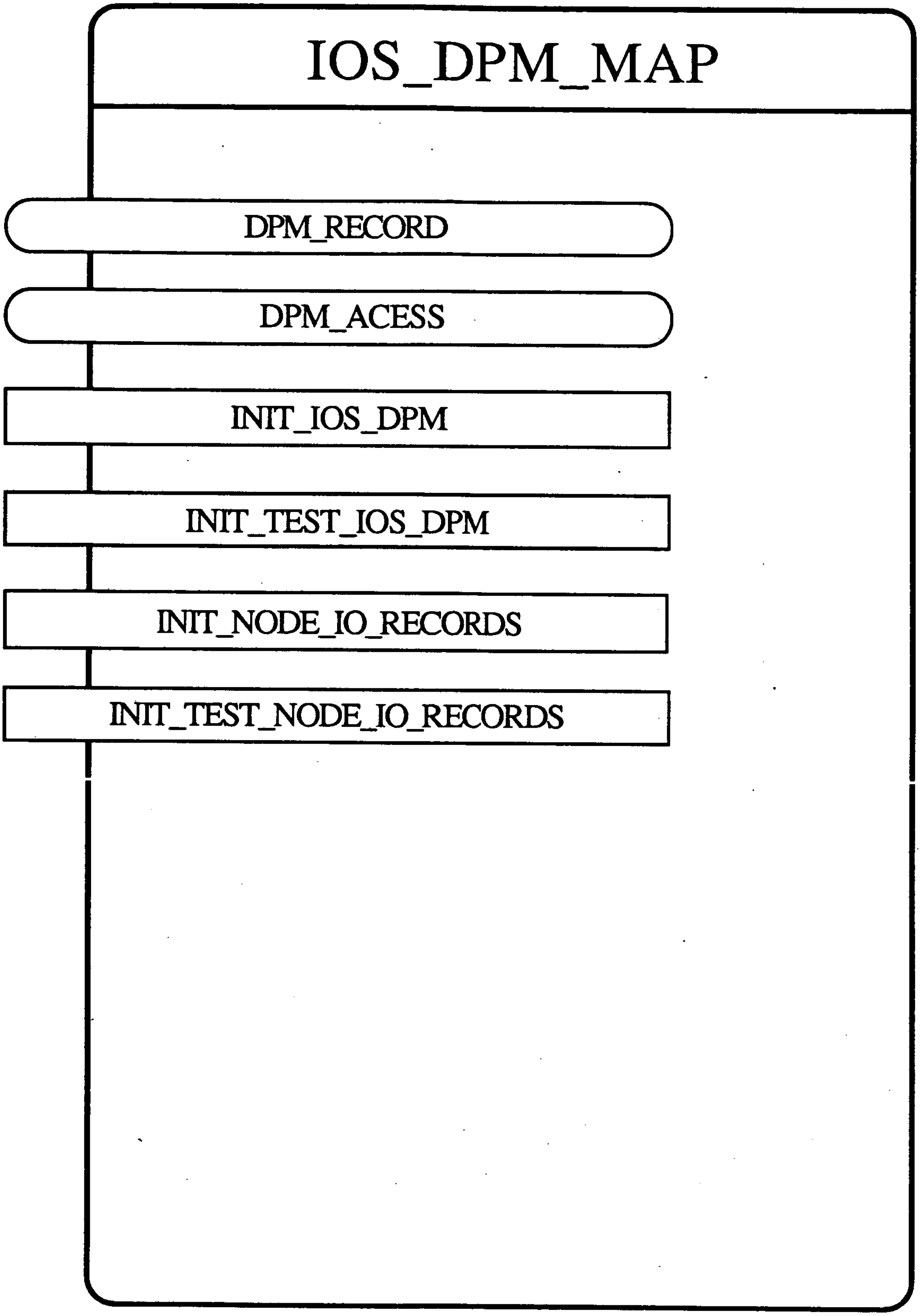
Subprogram: DPM_ADDR
Inputs: ios
channels
dpm_partition
rel_addr
Outputs dpm address

```
The high addr byte field of dpm address record is cleared.
The bus field of dpm record is assigned the constant shared bus.
The ios field of dpm record is assigned the value of the ios parameter.
The dpm_half field of  record is assigned the value of the dpm partition parameter.
The channel field of dpm record is assigned the value of the channels parameter.
The dpm_addr field of dpm record is asssigned the value of the rel addr parameter
return the dpm address
```

Subprogram: REL_DPM_ADDR
Inputs: system address
Outputs: relative dpm address

```
for < each CBA bit >
    Clear the bit
?? Are we using the upper DPM memory ??
no
yes
    Set high order bit of 12 bit address field.
    Clear bit 15
    (the bit that discriminates between lower and upper memory.)
return the low order word.
```

Software Specification Reference Number: 3.2.1.3



Software Specification Reference Number: 3.2.1.4

Subprogram: INIT_NODE_IO_RECORDS
Inputs network identifier
channel
Outputs: initialize status collection output transactions

```
Obtain topology and root link data from the local database.
for < each node in the network >
    Generate a node status command.
    Initialize output packet with the node status command.
    Initialize the output byte count for the configure nodes transaction
    Initialize the output byte count for the status transaction.
Branch from the last transaction to end of chain program.
```

Subprogram: INIT_TEST_NODE_IO_RECORDS
Inputs: network identifier
channel
Outputs: initialize ios test input and output transactions

```
Obtain topology and root_link data from the local database.
Generate the configure command to disable all ports in root node.
Initialize the first transaction with the above command.
Generate the configure command to enable root link port for 1 time.
Initialize the second transaction with the above command.
```

Software Specification Reference Number: 3.2.1.4

Subprogram: INIT_IOS_DPM
Inputs: dpm_ptr
Outputs: initializes ios, and the status collection and reconfiguration programs

```
Initialize the interface command register.
Zero dpm (non-register area).
Initialize the solicited chain pointer.
Initialize the unsolicited chain pointer to ios idle program.
Initialize timer limit register by disabling timer.
Initialize poll register number 1.
Disable hdlc autoflag mode.
Disable hdlc for transmission and reception.
Initialize end of chain program.
Initialize ios idle program.
Initialize node status program header.
for < the number of node transactions >
    Initialize node transaction instructions.
    ?? Is this the last node transaction ??
    no                              yes
    Branch to next transacton.      Branch to eoc program.
Initialize node config program header.
for < the number of node transactions >
    Initialize node transaction instructions.
    ?? Is this the last node transaction ??
    no                              yes
    Branch to next transacton.      Branch to eoc program.
```

Subprogram: INIT_TEST_IOS_DPM
Inputs: dpm_ptr
Outputs: initialize ios test program

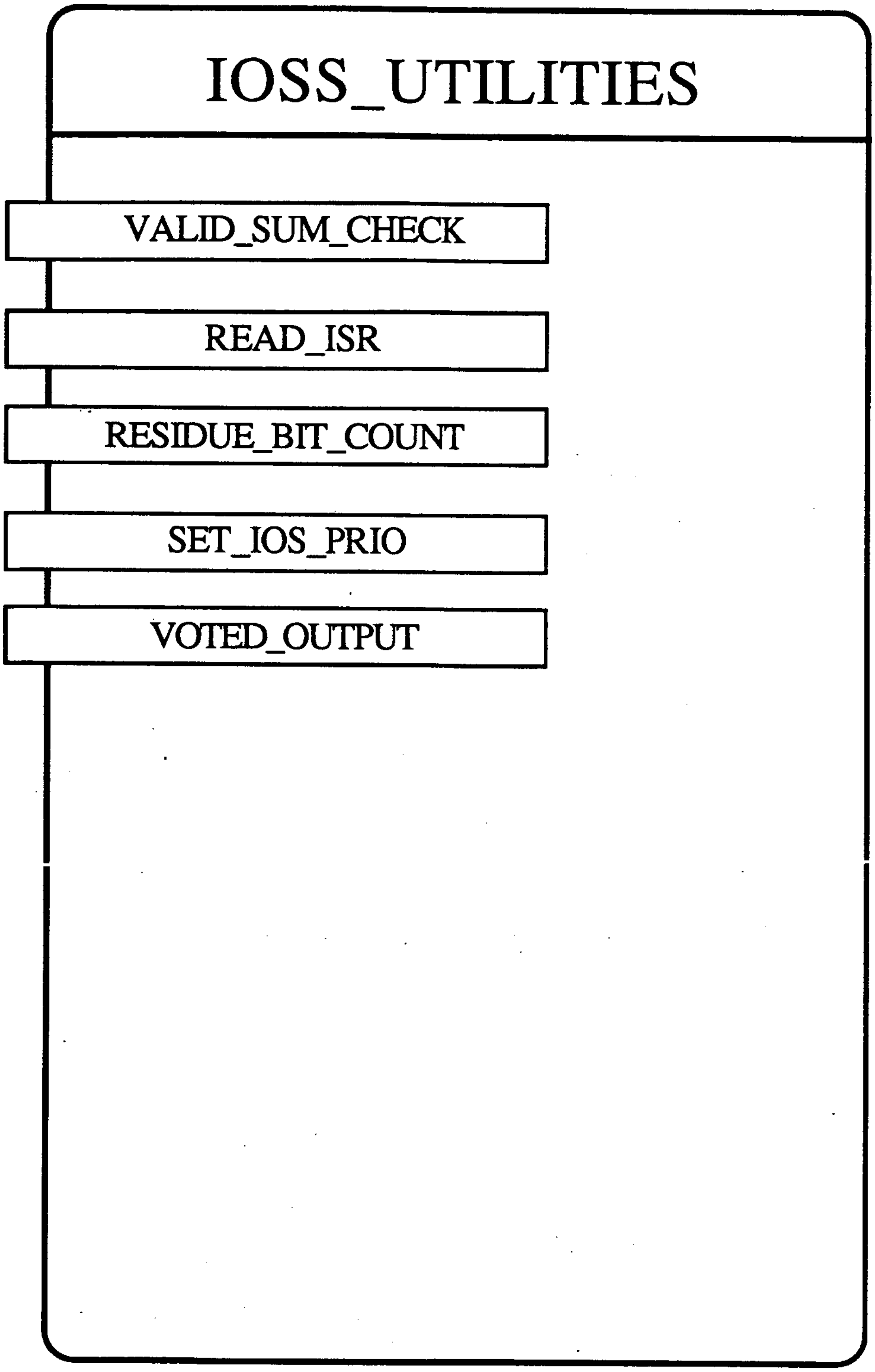
```
Initialize the header of the test program.
Initialize the first transaction instructions.
Initialize byte count of test input data to a non-zero value.
Initialize the second transaction instructions.
```

Software Specification Reference Number: 3.2.1.4

I/O Sequencer/Dual Ported Memory Map

REGION	FUNCTION	ADDRESS
0000-001F	HARDWARE REGISTERS:	0000
	Solicited Chain Ptr-R/W (Hi byte)	0001
	Solicited Chain Ptr-R/W (Lo byte)	0003
	Unsolicited Chain Ptr-R/W (Hi byte)	0004
	Unsolicited Chain Ptr-R/W (Lo byte)	0005
	Unused)	0006
	Chain Status Register (CSR)-R	0007
	Interface Command Register (ICR)-W	0008
	Interface Status Register (ISR)-R	0009
	Timer Limit Register (TLR)-W	0010
	Poll Register # 1 (PR1)-W	0011
	Poll Register # 2 (PR2)-W	0012
	Time-R	0013
	Reserved	0014
	Reserved	0015
	HDLC Control Register 1 (CR1)-R/W	0016
	HDLC Control Register 2 (CR2)-R/W	0017
	HDLC Control Register 3 (CR3)-R/W	0018
	HDLC Receiver Holding Register (RHR)-R	0019
	Address Register (AR)-W	001A
	HDLC Interrupt Register (IR)-R	001B
	Transmit Holding Register (THR)	001C
	HDLC Status Register (SR)-R	001D
	Reserved	001E
	Reserved	001F
0030-003F	CHAIN STATUS	
0040-00FF	INIT AND TEST PROGRAM	
0100-01FF	INIT AND TEST DATA	
0200-057F	NODE STATUS CHAIN	
0600-09FF	NODE CONFIGURATION CHAIN	
0A00-A7F	OUTPUT RECORDS FOR NODE STATUS CHAIN	
0A80-0AFF	OUTPUT RECORDS FOR NODE CONFIGURATION CHAIN	
0B00-0C7F	INPUT RECORDS FROM NODE STATUS CHAIN	
0C80-0DFF	INPUT RECORDS FROM NODE CONFIGURATION CHAIN	
0E00-0E7F	UNSOLICITED INPUT DATA BUFFFER	
0E80-0EFF	END OF CHAIN PROGRAM	
0F00-0FFF	UNSOLICITED CHAIN PROGRAM--KEEPS IOS IN IDLE	
1000-1FFF	USER CHAINS AND DATA	

Software Specification Reference Number: 3.2.1.4



Software Specification Reference Number: 3.2.1.5

Subprogram: VOTED_OUTPUT
Inputs: value
address
Outputs: none

```
Perform a from_all exchange on value.
Write voted value to address
```

Subprogram: VALID_SUM_CHECK
Inputs: byte_ptr
byte_count
Outputs: boolean flag

```
Temp_sum is assigned a zero.
for < the number of bytes >
    Temp_sum is assigned temp_sum plus the next byte
    ?? Is temp_sum less than or equal to modulus ??
    no
    yes
        Subtract modulus from temp_sum.
?? Is temp_sum equal to zero ??
no
    return a FALSE
yes
    return a TRUE
```

Subprogram: RESIDUE_BIT_COUNT
Inputs: status register
Outputs: number of residue bits

```
The status register is converted into a hdlc_cr3_type.
return the residue_bit_count field of cr3.
```

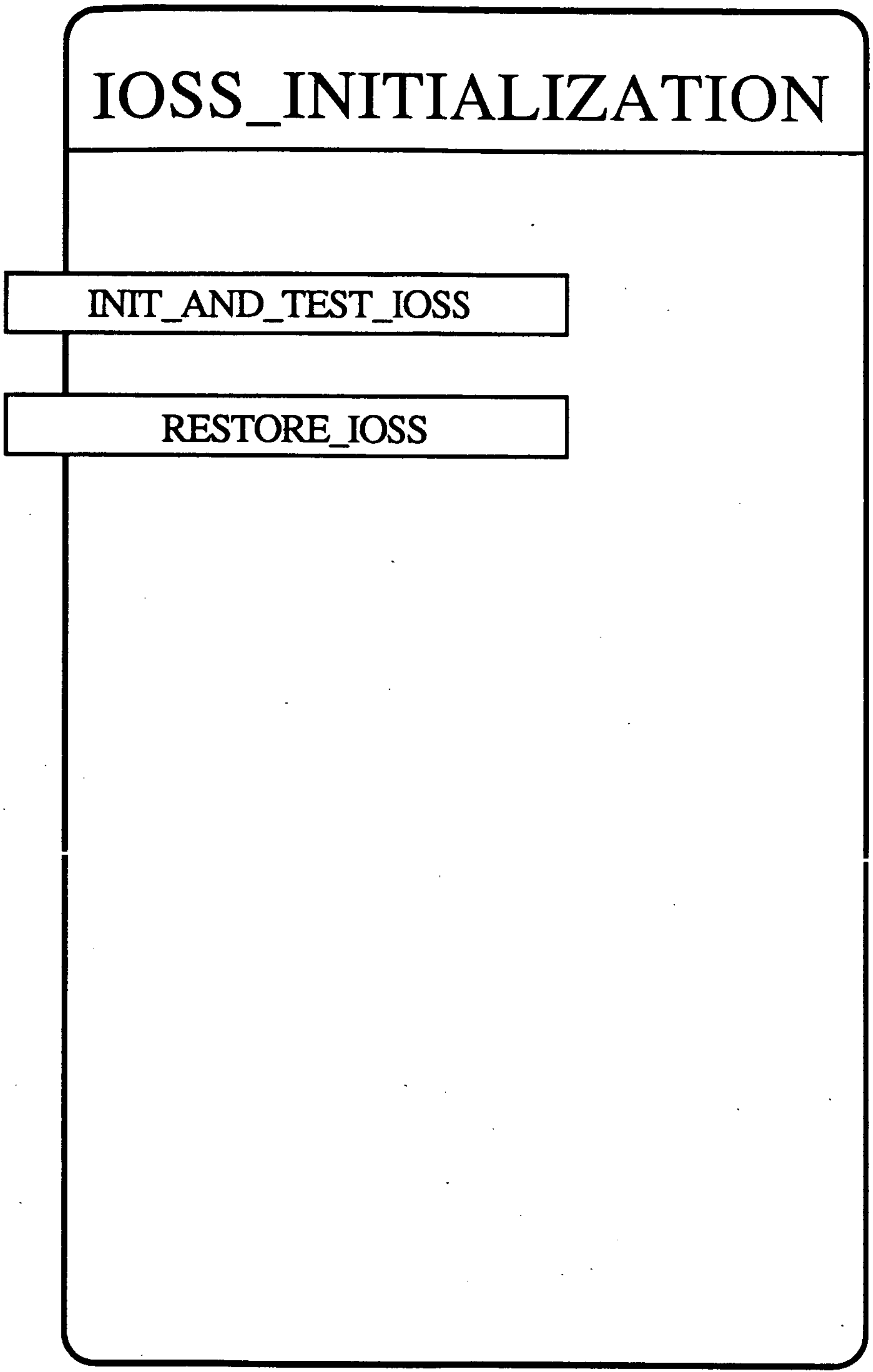
Subprogram: SET_IOS_POLL_PRIO
Inputs: prio
Outpus: ios instruction

```
The level field of PR2 is set to ios_poll_level.
The prio field of PR2 is set to the passed in prio.
The instruction is set up.
return the instruction
```

Subprogram: READ_ISR
Inputs: none
Outputs: value of the interface status register

```
Read the location addressed as the ICR
Type cast this value as interface status register type
Return converted value
```

Software Specification Reference Number: 3.2.1.5



Software Specification Reference Number: 3.2.2

Subprogram: INIT_AND_TEST_IOSS
Inputs: interface status
active
Outputs: none

```
for < each network identifier >
    ?? Is this network connected to my gpc ??
    no
    yes
        Read the network status from i/o network status
        Stop the ios
        Perform the dpm address line test
        Perform the dpm memory read/write test
        Perform the network interface self test
        ?? Are there any non-failed ios's
        no
        yes
            Set active to TRUE
        Write update interface status to IO network status
```

Subprogram: RESTORE_IOSS
Inputs: root links
dpm pointer
Outputs: channel to be restored
updated status collection chain
initialized ios

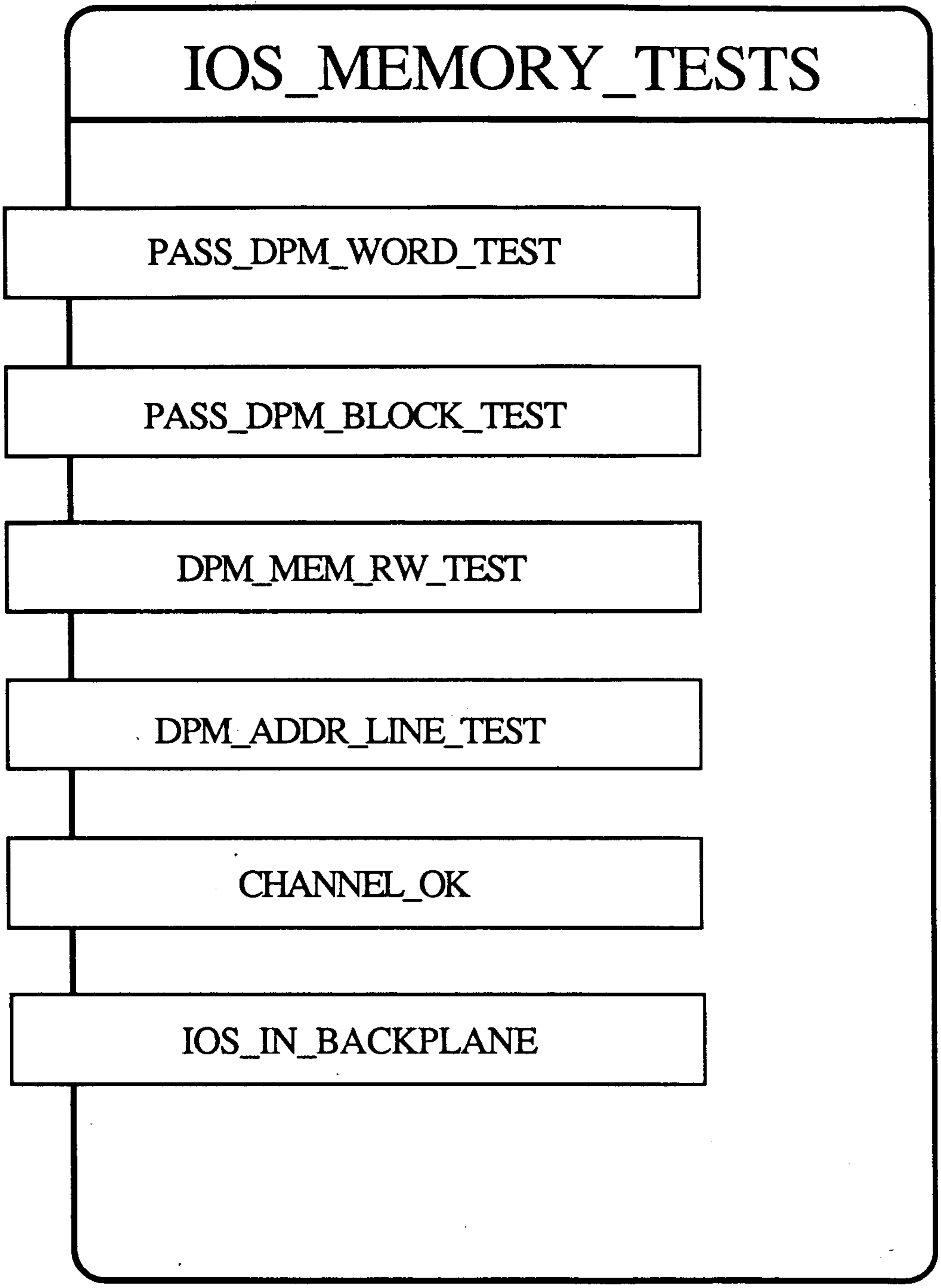
```
for < each network identifier >
    ?? Is this network connected to my gpc ??
    no
    yes
        Retrieve the root links from the local database
        for < each root link that is connected to channel to be restored >
            Initialize the registers and program chains
            Initialize the node i/o data
            update node status collection program
```

Software Specification Reference Number: 3.2.2

Subprogram: NETWORK_INTERFACE_SELF_'TEST
Inputs: network identifier
interface_status
Outputs: none

```
Initialize DPMS
for < the number of channels >
    ?? Does GPC_FDIR report channel failed ??
    yes:
        mark the interface status failed channel
    no:
        Run chain without poll to shut off root node and disable possible babbler
        Run ios self test chain
        Wait for chain to complete
        Read csr, isr, test data, final csr from dpm
        ?? Did channel fail during the test ??
        yes:
            mark the interface status failed channel
        no:
            ?? ios program not complete ??
            yes:
                mark the interface status failed ios and log the error
            no:
                ?? csr fail to reset ??
                yes
                no:
                    ?? isr get stuck on high ??
                    yes
                    no:
                        ?? transmisson error in root node ??
                        yes
                        no:
                            ?? timeout test fail ??
                            yes
                            no:
                                ?? transmisson error in root port ??
                                yes
                                no:
                                    ?? input byte count incorrect ??
                                    yes
                                    no:
                                        ?? HDLC protocol errors ??
                                        yes
                                        no:
                                            ?? residual bit count incorrect ??
                                            yes
                                            no:
                                                ?? checksum invalid ??
                                                yes
                                                no:
                                                    ?? node address incorrect ??
                                                    yes
                                                    no:
                                                        ?? final chain csr invalid ??
                                                        yes
                                                        no
```

Software Specification Reference Number: 3.2.2, 3.2.3.2



Software Specification Reference Number: 3.2.3.1

Subprogram: PASS_DPM_WORD_TEST
Inputs: network identifier
channel
Outputs: boolean flag

<table>
<tr><td colspan="3">Store the word to be tested in a tempory variable</td></tr>
<tr><td colspan="3">Fill the word with pattern 1</td></tr>
<tr><td colspan="3">?? Does the word contain pattern 1 ??</td></tr>
<tr><td>no</td><td colspan="2">yes</td></tr>
<tr><td rowspan="4">log an error failed pattern test</td><td colspan="2">Fill the word with pattern 2</td></tr>
<tr><td colspan="2">?? Does the word contain pattern 2 ??</td></tr>
<tr><td>no</td><td>yes</td></tr>
<tr><td>restore the word</td><td>restore the word</td></tr>
<tr><td>restore the word</td><td>log an error message pattern test</td><td>calculate the offset for the next call to this routine</td></tr>
<tr><td>return a FALSE</td><td>return a FALSE</td><td>return a TRUE</td></tr>
</table>

Subprogram: PASS_DPM_BLOCK_TEST
Inputs: network identifier
channel
start address
end address
Outputs: boolean flag

<table>
<tr><td colspan="4">Store the start and end addresses in a tempory variable</td></tr>
<tr><td colspan="4">?? Is the start address greater than or equal to the end address ??</td></tr>
<tr><td>yes</td><td colspan="3">no</td></tr>
<tr><td rowspan="6">log an error message invalid address range</td><td colspan="3">Ensure that the start and end adresses are divisable by two</td></tr>
<tr><td colspan="3">for < the number of words ></td></tr>
<tr><td colspan="3">Store the current word in a tempory variable</td></tr>
<tr><td colspan="3">Fill the current word with pattern 1</td></tr>
<tr><td colspan="3">?? Does the current word contain pattern 1 ??</td></tr>
<tr><td>no</td><td colspan="2">yes</td></tr>
<tr><td rowspan="6">return a TRUE</td><td rowspan="5">log an error failed pattern test</td><td colspan="2">Fill the current word with pattern 2</td></tr>
<tr><td colspan="2">?? Does the current word contain pattern 2 ??</td></tr>
<tr><td>yes</td><td>no</td></tr>
<tr><td>restore the word</td><td>restore the word</td></tr>
<tr><td rowspan="2">return a TRUE</td><td>log an error failed pattern test</td></tr>
<tr><td>return a FALSE</td><td>return a FALSE</td></tr>
</table>

Software Specification Reference Number: 3.2.3.1

Subprogram: DPM_MEM_RW_TEST
Inputs: network identifier
Outputs: interface status
interface status

```
Write test pattern 1 into memory
Verify with test pattern 1
Write test pattern 2 into memory
Verify with test pattern 2
```

Subprogram: WRITE
Inputs: test pattern
Outputs: none

```
Write test pattern in lower partition of memory
Write test pattern in upper partition of memory
```

Subprogram: VERIFY
Inputs: test pattern
Outputs: none

```
for < the number of channels >
    ?? Is the interface status of this channel ok ??
    no
    yes
        ?? Has the channel failed according to the GPCFDIR ??
        yes
            log an error
            Channel failed
            before test.
            mark interface status failed channel
        no
            Initialize failure to FALSE
            for < each location in the lower partition of memory >
                ?? Does the pattern match the pattern expected ??
                yes
                no
                    Set failure to TRUE
                    Save the address where the failure occured
                    ?? Did the channel fail ??
                    no
                    yes
                        log an error message
                        channel failed after test
                        mark interface status failed channel
                    exit
            for < each location the upper partition of memory >
                ?? Does the pattern match the pattern expected ??
                yes
                no
                    Set failure to TRUE
                    Save the address where the failure occured
                    ?? Did the channel fail ??
                    no
                        log an error message
                        IOS failed
                        mark interface status
                        failed IOS
                    yes
                        log an error message
                        channel failed after test
                        mark interface status
                        failed channel
                    exit
```

Software Specification Reference Number: 3.2.3.1

Subprogram: DPM_ADDR_LINE_TEST
Inputs: network identifier
interface status
Outputs: interface status

```
Test with normal pattern by:
    Write pattern to corresponding lower memory partition location
    Write pattern to corresponding upper memory partition location
    Read back the pattern from each channel by:
        for < each channel >
            ?? Is the interface status of this channel ok ??
            no
            yes
                ?? Has the channel failed according to the GPC_FDIR ??
                yes
                    log an error
                    Channel failed before test.
                    mark interface status failed channel
                no
                    Initialize failure to FALSE
                    for < each location in the lower partition of memory >
                        ?? Does the pattern match the pattern expected ??
                        yes
                        no
                            Set failure to TRUE
                            Save the address where the failure occured and exit
                    for < each location in the upper partition of memory >
                        ?? Does the pattern match the pattern expected ??
                        yes
                        no
                            Set failure to TRUE
                            Save the address where the failure occured and exit
                    ?? Does Failure equal TRUE ??
                    no
                    yes
                        ?? Did the channel fail ??
                        no
                            log an error
                            IOS Failed
                            mark interface status failed IOS
                        yes
                            log an error
                            Channel Failed After Test.
                            mark interface status failed channel
```

Software Specification Reference Number: 3.2.3.1

Subprogram: DPM_ADDR_LINE_TEST (Continued)
Inputs: network identifier
interface status
Outputs: interface status

```
Test with juxtaposed pattern by::
Write pattern to corresponding lower memory partition location with juxtaposed bytes.
Write pattern to corresponding upper memory partition location with juxtaposed bytes.
Read back the pattern from each channel by:
for < each channel >
    ?? Is the interface status of this channel' ok ??
    no
    yes
        ?? Has the channel failed according to the GPC_FDIR ??
        yes
            log an error message Channel failed before test.
            mark inter-face status failed channel
        no
            Initialize failure to FALSE
            for < each location in the lower partition of memory >
                ?? Does the pattern match the pattern expected ??
                yes
                no
                    Set failure to TRUE
                    Save the address where the failure occured and exit
            for < each location in the upper partition of memory >
                ?? Does the pattern match the pattern expected ??
                yes
                no
                    Set failure to TRUE
                    Save the address where the failure occured and exit
            ?? Does Failure equal TRUE ??
            no
            yes
                ?? Did the channel fail ??
                no
                    log an error
                    IOS Failed
                    mark interface status
                    failed IOS
                yes
                    log an error
                    Channel Failed After Test.
                    mark interface status
                    failed channel
```

Software Specification Reference Number: 3.2.3.1

Subprogram: CHANNEL_OK
Inputs: channel identifier
Outputs: boolean flag

```
?? case < channel identifier > ??
    channel A
        transmit pattern 1 from A
        assign check exchanged value
    channel B
        transmit pattern 1 from B
        assign check exchanged value
    channel C
        transmit pattern 1 from C
        assign check exchanged value
?? Does check equal pattern 1 ??
    no
        return FALSE
    yes
        ?? case < channel identilfier > ??
            channel A
                transmit pattern 2 from A
                assign check exchange value
            channel B
                transmit pattern 2 from B
                assign check exchange value
            channel C
                transmit pattern 2 from C
                assign check exchange value
        ?? Does check equal pattern 2 ??
            no
                return a FALSE
            yes
                return a TRUE
```

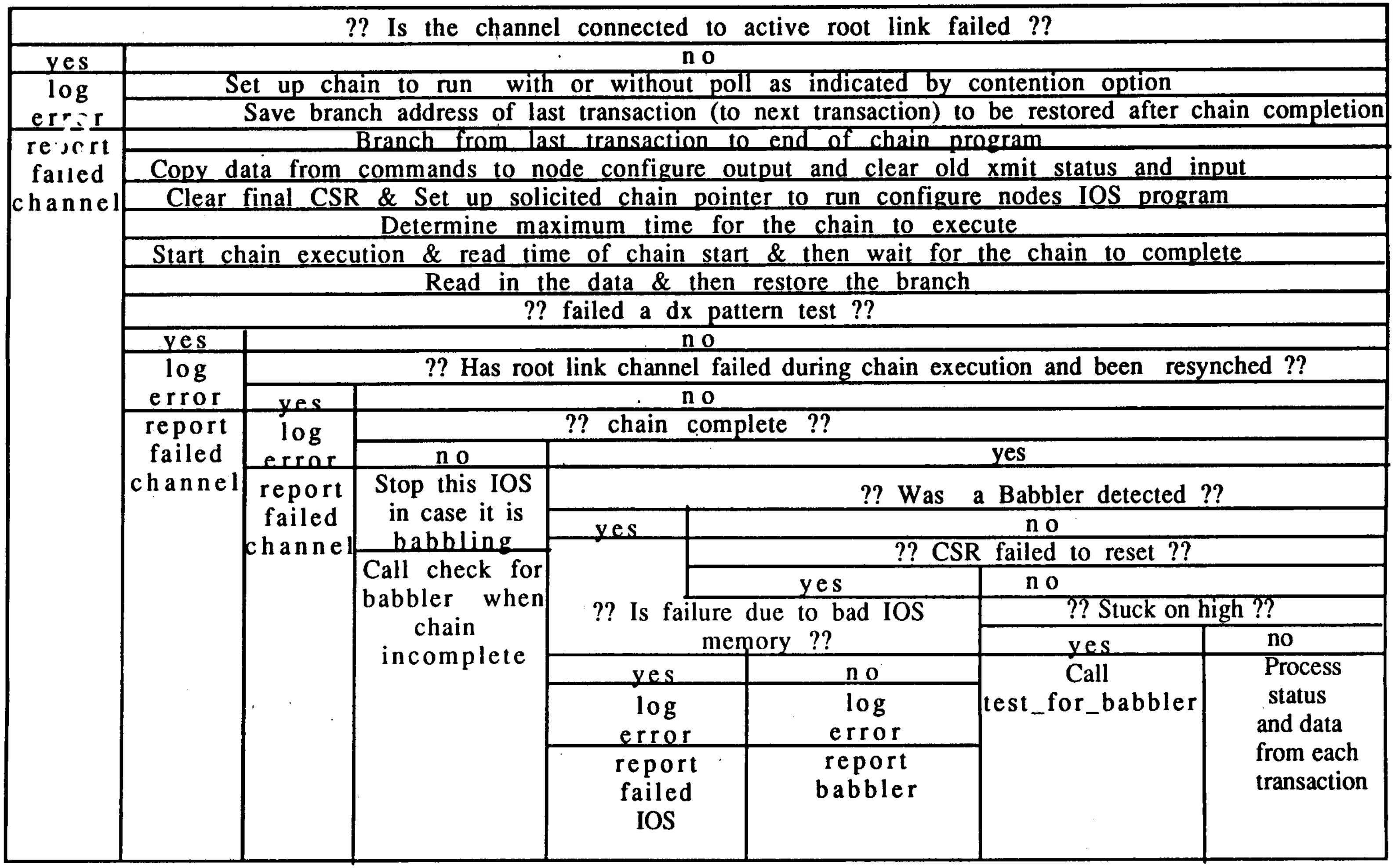
Subprogram: IOS IN BACKPLANE
Inputs: network identifier
Outputs: none

```
Write a pattern to the solicited chain pointer of the IOS
For < each root link in the network >
    ?? Does the pattern match the pattern written ??
        yes
        no
            The ios is deemed unreachable by the FTP and the error is logged
            The interface status of the ios is marked failed_ios
```

Software Specification Reference Number: 3.2.3.1

Software Specification Reference Number: 3.2.4

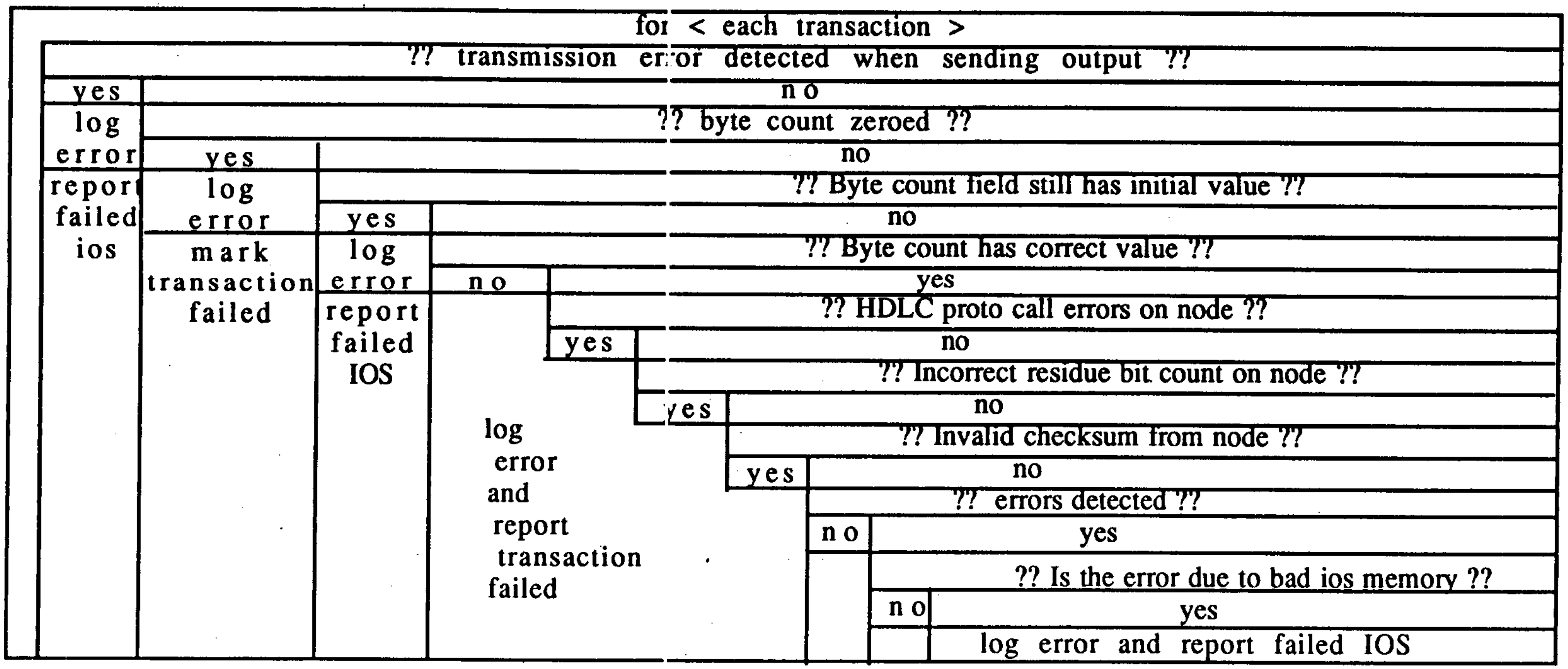
Subprogram: CONFIGURE_NODES
Inputs: i/o network identifier
active root link
configuration commands
contention option
Outputs: configuration report



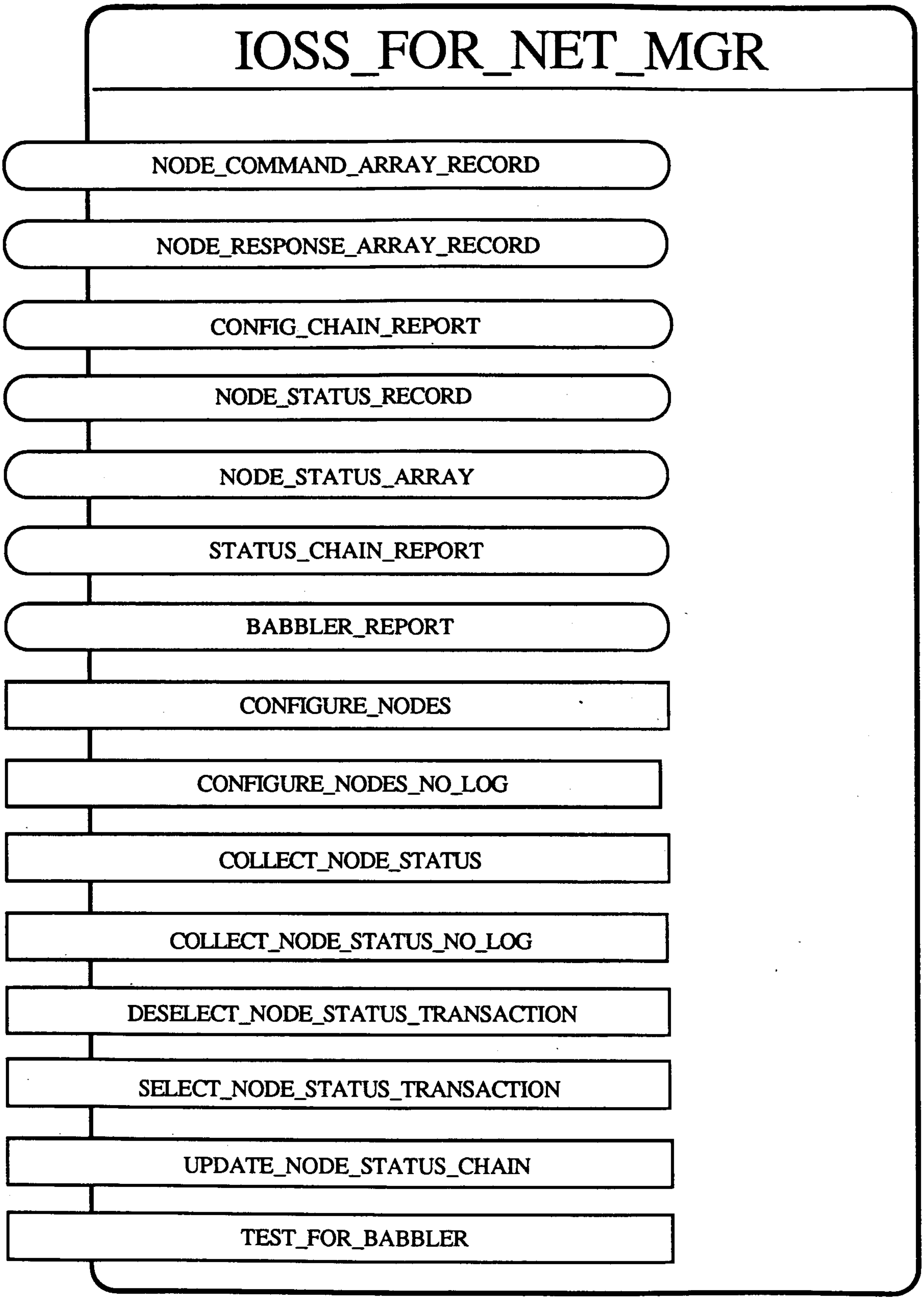
```
?? Is the channel connected to active root link failed ??
yes:
    log error
    report failed channel
no:
    Set up chain to run with or without poll as indicated by contention option
    Save branch address of last transaction (to next transaction) to be restored after chain completion
    Branch from last transaction to end of chain program
    Copy data from commands to node configure output and clear old xmit status and input
    Clear final CSR & Set up solicited chain pointer to run configure nodes IOS program
    Determine maximum time for the chain to execute
    Start chain execution & read time of chain start & then wait for the chain to complete
    Read in the data & then restore the branch
    ?? failed a dx pattern test ??
    yes:
        log error
        report failed channel
    no:
        ?? Has root link channel failed during chain execution and been resynched ??
        yes:
            log error
            report failed channel
        no:
            ?? chain complete ??
            no:
                Stop this IOS in case it is babbling
                Call check for babbler when chain incomplete
            yes:
                ?? Was a Babbler detected ??
                yes:
                    ?? Is failure due to bad IOS memory ??
                    yes:
                        log error
                        report failed IOS
                    no:
                        log error
                        report babbler
                no:
                    ?? CSR failed to reset ??
                    yes:
                        (?? Is failure due to bad IOS memory ??)
                    no:
                        ?? Stuck on high ??
                        yes:
                            Call test_for_babbler
                        no:
                            Process status and data from each transaction
```

Software Specification Reference Number: 3.2.4.1

Subprogram: PROCESS STATUS AND DATA FROM NODE TRANSACTIONS
Inputs: node input packets
Outputs: configuration report


for < each transaction >
?? transmission er:or detected when sending output ??
yes
log error
report failed ios
no
?? byte count zeroed ??
yes
log error
mark transaction failed
no
?? Byte count field still has initial value ??
yes
log error
report failed IOS
no
?? Byte count has correct value ??
no
log error and report transaction failed
yes
?? HDLC proto call errors on node ??
yes
no
?? Incorrect residue bit count on node ??
yes
no
?? Invalid checksum from node ??
yes
no
?? errors detected ??
no
yes
?? Is the error due to bad ios memory ??
no
yes
log error and report failed IOS

Software Specification Reference Number: 3.2.4.1

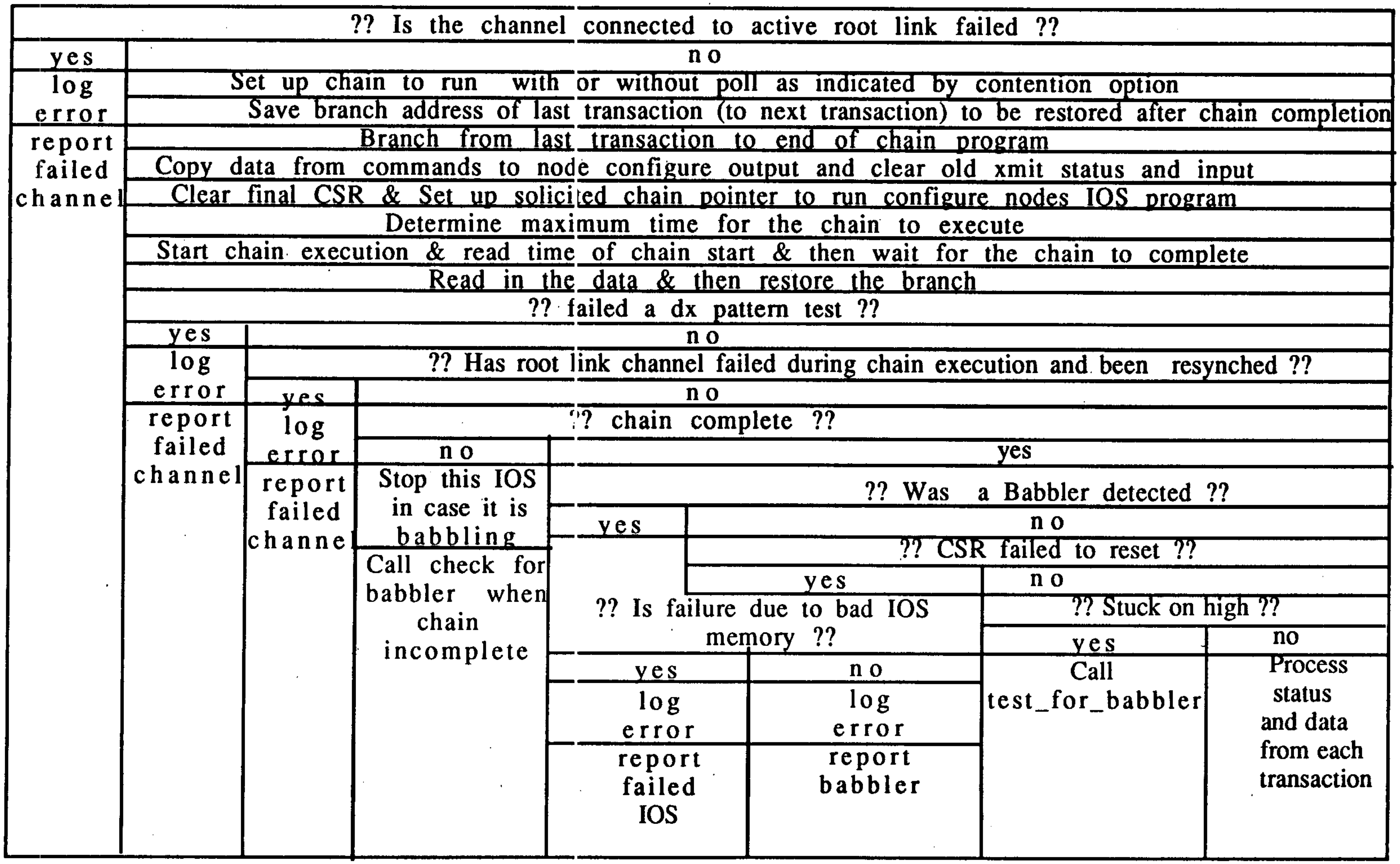


Software Specification Reference Number: 3.2.4

Subprogram: CONFIGURE_NODES

Inputs:
- i/o network identifier
- active root link
- configuration commands
- contention option

Outputs:
- configuration report



```
?? Is the channel connected to active root link failed ??
  yes
    log error
    report failed channel
  no
    Set up chain to run with or without poll as indicated by contention option
    Save branch address of last transaction (to next transaction) to be restored after chain completion
    Branch from last transaction to end of chain program
    Copy data from commands to node configure output and clear old xmit status and input
    Clear final CSR & Set up solicited chain pointer to run configure nodes IOS program
    Determine maximum time for the chain to execute
    Start chain execution & read time of chain start & then wait for the chain to complete
    Read in the data & then restore the branch
    ?? failed a dx pattern test ??
      yes
        log error
        report failed channel
      no
        ?? Has root link channel failed during chain execution and been resynched ??
          yes
            log error
            report failed channel
          no
            ?? chain complete ??
              no
                Stop this IOS in case it is babbling
                Call check for babbler when chain incomplete
              yes
                ?? Was a Babbler detected ??
                  yes
                    ?? Is failure due to bad IOS memory ??
                      yes
                        log error
                        report failed IOS
                      no
                        log error
                        report babbler
                  no
                    ?? CSR failed to reset ??
                      yes
                        ?? Is failure due to bad IOS memory ??
                          yes
                            log error
                            report failed IOS
                          no
                            log error
                            report babbler
                      no
                        ?? Stuck on high ??
                          yes
                            Call test_for_babbler
                          no
                            Process status and data from each transaction
```

Software Specification Reference Number: 3.2.4.1

Subprogram: CHECK_FOR_BABLER_WHEN_CHAIN_INCOMPLETE
Inputs: final csr
Outputs: boolean flag

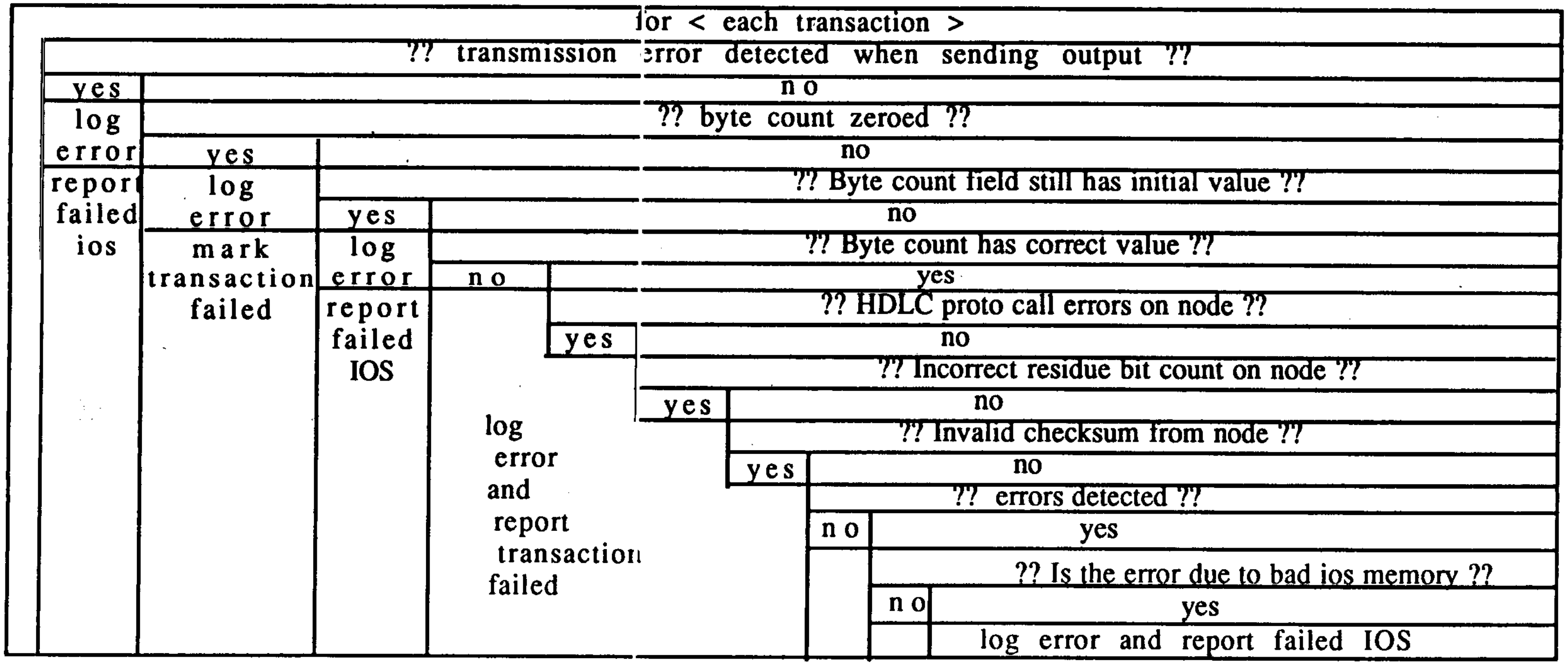
```
?? chain executed with contention ??
  no
  yes
    ?? attempted poll incomplete or bus stuck high or data bit received
       during poll or poll detected during data transmission ??
      no
      yes
        log error and report a babbler found
?? Above logic detected babbler ??
  yes
  no
    ?? SCP indicates poll command still being executed ??
      yes
        log error
        report babble found
      no
        ?? Branch to EOC not made to SCP or Switch to unsolicited chain not made by SCP ??
          yes
            log erro
            report failed IOS
          no
            for << each transaction >>
              ?? transaction completed & byte count holds initial value or transaction not complete ??
                no
                yes
                  report a failed IOS
    ?? Any failures found ??
      no
      yes
        ??Is failure due to bad IOS memory ??
          no
          yes
            log error and report failed IOS
```

Subprogram: BABBLER_DETECTED_WHEN_CHAIN_COMPLETE
Inputs: final csr
Outputs: boolean flag

```
?? Data bit received during message transmission ??
  yes
    log error
    report a babbler found
  no
    ?? Chain executed with contention ??
      no
      yes
        ?? Data bit received during poll or poll detected during message transmission ??
          no
          yes
            log error and report a babbler found
```

Software Specification Reference Number: 3.2.4.1

Subprogram: PROCESS STATUS AND DATA FROM NODE TRANSACTIONS
Inputs: node input packets
Outputs: configuration report



```
for < each transaction >
    ?? transmission error detected when sending output ??
    yes
        log error
        report failed ios
    no
        ?? byte count zeroed ??
        yes
            log error
            mark transaction failed
        no
            ?? Byte count field still has initial value ??
            yes
                log error
                report failed IOS
            no
                ?? Byte count has correct value ??
                no
                    log error and report transaction failed
                yes
                    ?? HDLC proto call errors on node ??
                    yes
                    no
                        ?? Incorrect residue bit count on node ??
                        yes
                        no
                            ?? Invalid checksum from node ??
                            yes
                            no
                                ?? errors detected ??
                                no
                                yes
                                    ?? Is the error due to bad ios memory ??
                                    no
                                    yes
                                        log error and report failed IOS
```

Software Specification Reference Number: 3.2.4.1

Subprogram: COLLECT_STATUS
Inputs: i/o network identifier
active root link
Outputs: configuration report

```
?? Is a channel connected to active root link failed ??
  yes
    log error
    report failed channel
  no
    Clear old xmit status and input
    Clear final CSR
    Set up solicited chain pointer to run configure nodes IOS program
    Determine maximum time for the chain to execute
    Start chain execution
    Read time of chain start
    Wait for the chain to complete
    Read in the data
    ?? failed a dx pattern test ??
      yes
        log error
        report failed channel
      no
        ?? Has root link channel failed during chain execution and been resynched ??
          yes
            log error
            report failed channe
          no
            ?? chain complete ??
              no
                Stop this IOS in case it is babbling
                Call check for babbler
              yes
                ?? CSR failed to reset ??
                  yes
                    ?? Is failure due to bad IOS memory ??
                      yes
                        log error
                        report failed IOS
                      no
                        log error
                        report babbler
                  no
                    ?? Stuck on high ??
                      yes
                        Call test_for_babbler
                      no
                        Process status and data from status collection
```

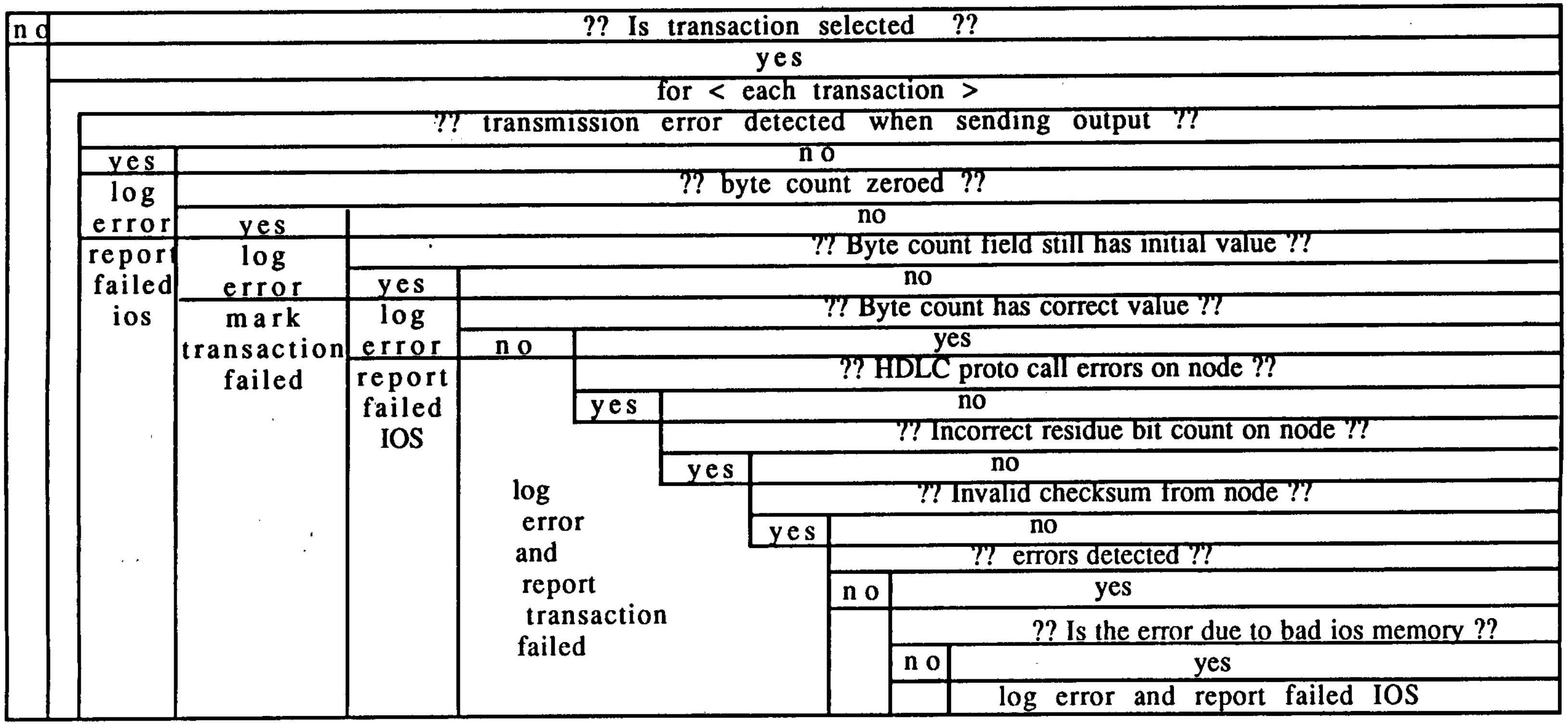
Software Specification Reference Number: 3.4.2.2

Subprogram: CHECK_FOR_BABLER_DURING_STATUS_COLLECTION
Inputs: final csr
Outputs:. boolean flag

```
?? Attempted poll incomplete or bus stuck high or data bit
recieved during poll or poll detected durin data transmission ??
  yes
    log error
    report babbled during contention
  no
    ?? Babbled during transmission ??
      yes
        log error
        report babbled during transmision
      no
        ?? IOSPg did not compute SCP ??
          yes
            log error
            report babbled during contention
          no
            ?? Branch to EOC not made SCP ??
              yes
                log error
                report failed IOS
              no
                ?? Switch to unsolicited not made SCP ??
                  yes
                    log error
                    report failed IOS
                  no
                    for << each transaction >>
                      ?? transaction completed &
                      byte count holds initial value
                      or transaction not complete ??
                        no
                        yes
                          report a failed IOS
?? Any failures found ??
  no
  yes
    ?? Is failure due to bad IOS memory ??
      no
      yes
        Log error and report failed IOS
```

Software Specification Reference Number: 3.4.2.2

Subprogram: PROCESS STATUS AND DATA FROM STATUS COLLECTION
Inputs: node input packets
Outputs: configuration report



```
?? Is transaction selected ??
no
yes
    for < each transaction >
        ?? transmission error detected when sending output ??
        yes
            log error
            report failed ios
        no
            ?? byte count zeroed ??
            yes
                log error
                mark transaction failed
            no
                ?? Byte count field still has initial value ??
                yes
                    log error
                    report failed IOS
                no
                    ?? Byte count has correct value ??
                    no
                        log error and report transaction failed
                    yes
                        ?? HDLC proto call errors on node ??
                        yes
                        no
                            ?? Incorrect residue bit count on node ??
                            yes
                            no
                                ?? Invalid checksum from node ??
                                yes
                                no
                                    ?? errors detected ??
                                    no
                                    yes
                                        ?? Is the error due to bad ios memory ??
                                        no
                                        yes
                                            log error and report failed IOS
```

Software Specification Reference Number: 3.4.2.2

Subprogram: DESELECT_NODE_STATUS_TRANSACTION
Inputs: network identifier
node
Outputs: none

```
?? Is the node currently selected ??
no
yes
    Set the node selection status to FALSE
    Decrement the active node count
    Set up pointer to all DPMs connected to network
    ?? Is a previous node selected ??
    no
        Branch from header
    yes
        Branch from the previous node
    ?? Is a subsequent node selected ??
    no
        Branch to end of chain
    yes
        Branch to that node
```

Subprogram: SELECT_NODE_STATUS_TRANSACTION
Inputs: network identifier
node
Outputs: none

```
?? Is the node currently deselected ??
no
yes
    Set up pointer to all DPMs connected to network
    ?? Is a previous node selected ??
    no
        Branch from header to selected transaction
    yes
        Branch from a previous node to selected transaction
    ?? Is a subsequent node selected ??
    no
        Branch from selected transaction to end of chain
    yes
        Branch to that node from selected transaction
    Set the node selection status to TRUE
    Increment the active node count
```

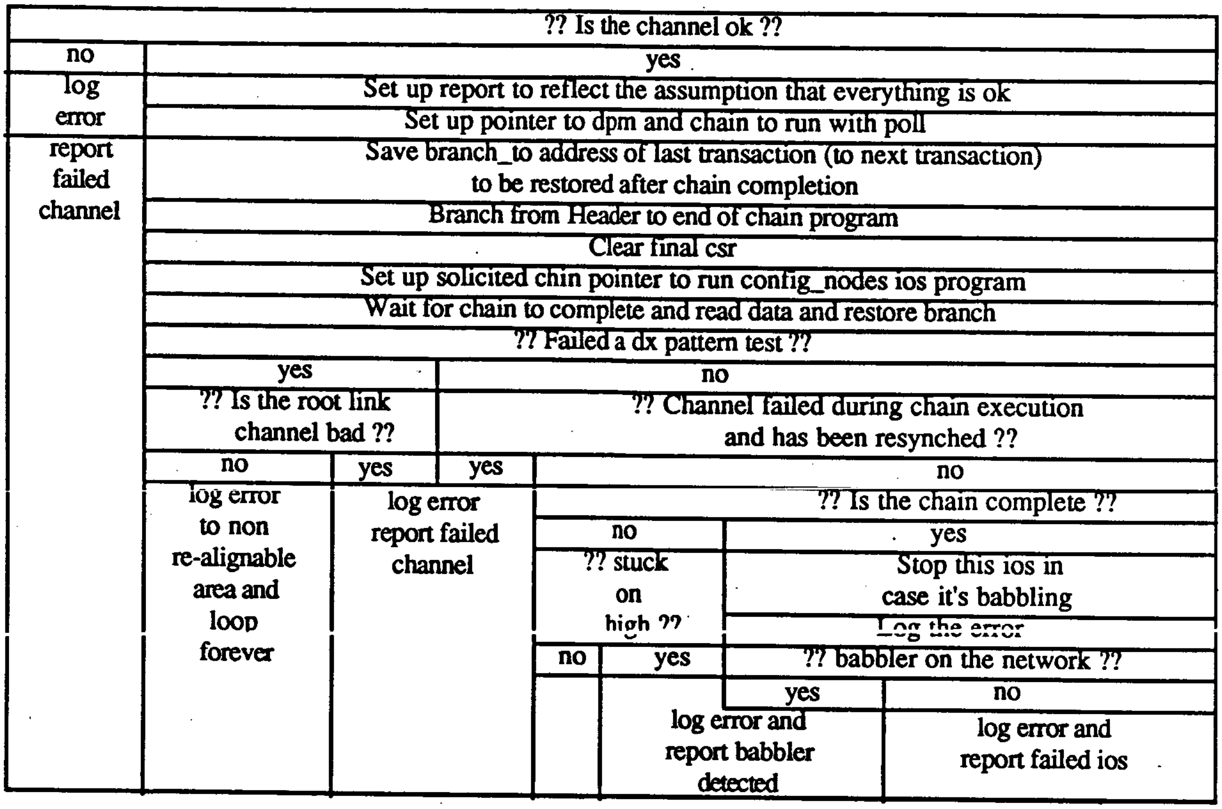
Software Specification Reference Number: 3.4.2.3

Subprogram: UPDATE_NODE_STATUS_CHAIN
Inputs: network identifier
channel
Outputs: none

```
Set up pointers to the DPM designated by network identifier and channel
Set found to FALSE
for << each node >>
    ?? Is the status transaction for this node selected ??
    no
    yes
        Set found to TRUE
        ?? Is this the first selected transaction ??
        no
            Branch to this transaction
            from previous transaction
        yes
            Branch to this transaction
            from header
        Set previous transaction to the current transaction
??Any transactions selected ??
no
yes
    Branch from last selected transaction to end of chain program
```

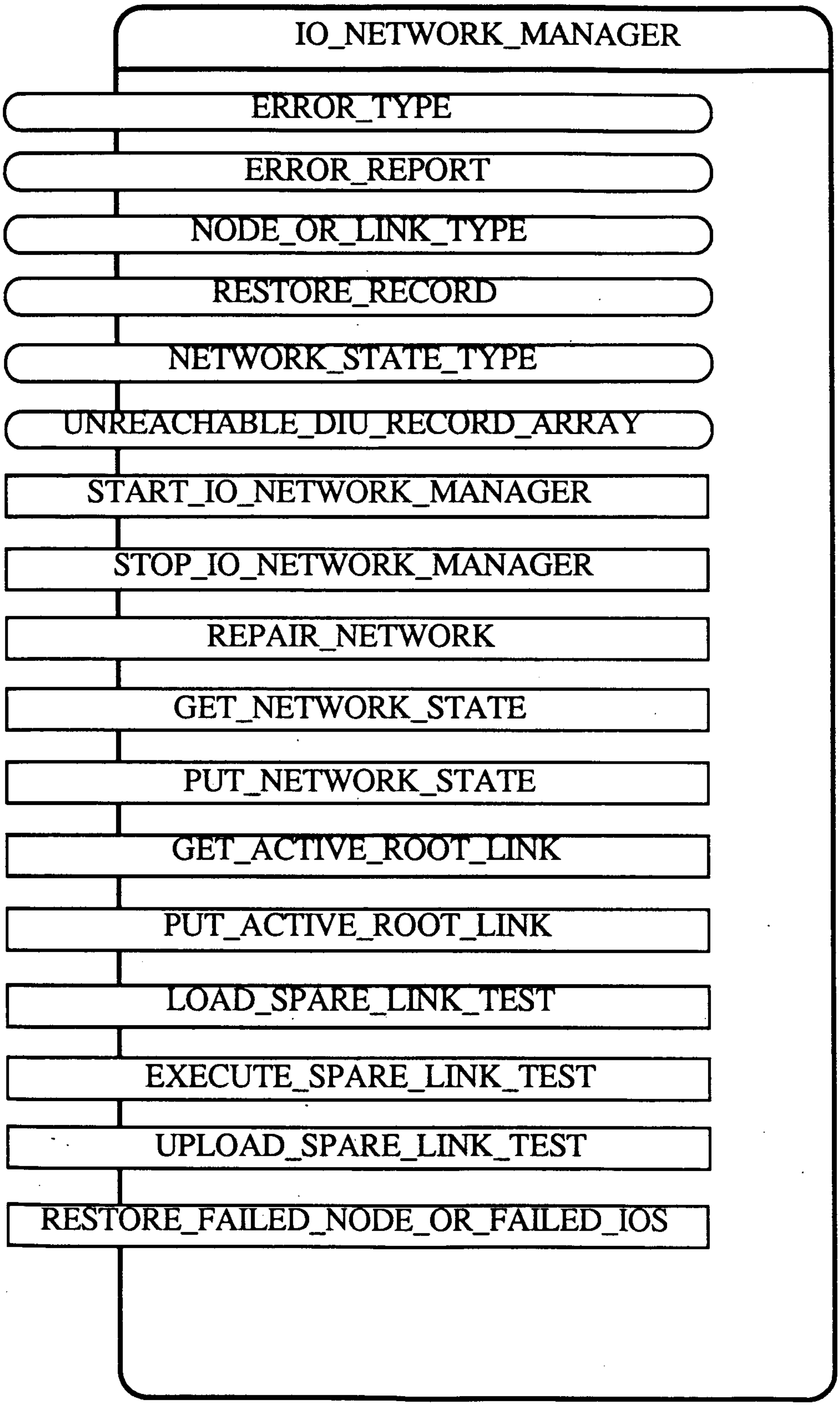
Software Specification Reference Number: 3.4.2.3

Subprogram: TEST_FOR_BABBLER
Inputs: network identifier
root link
Outputs: report

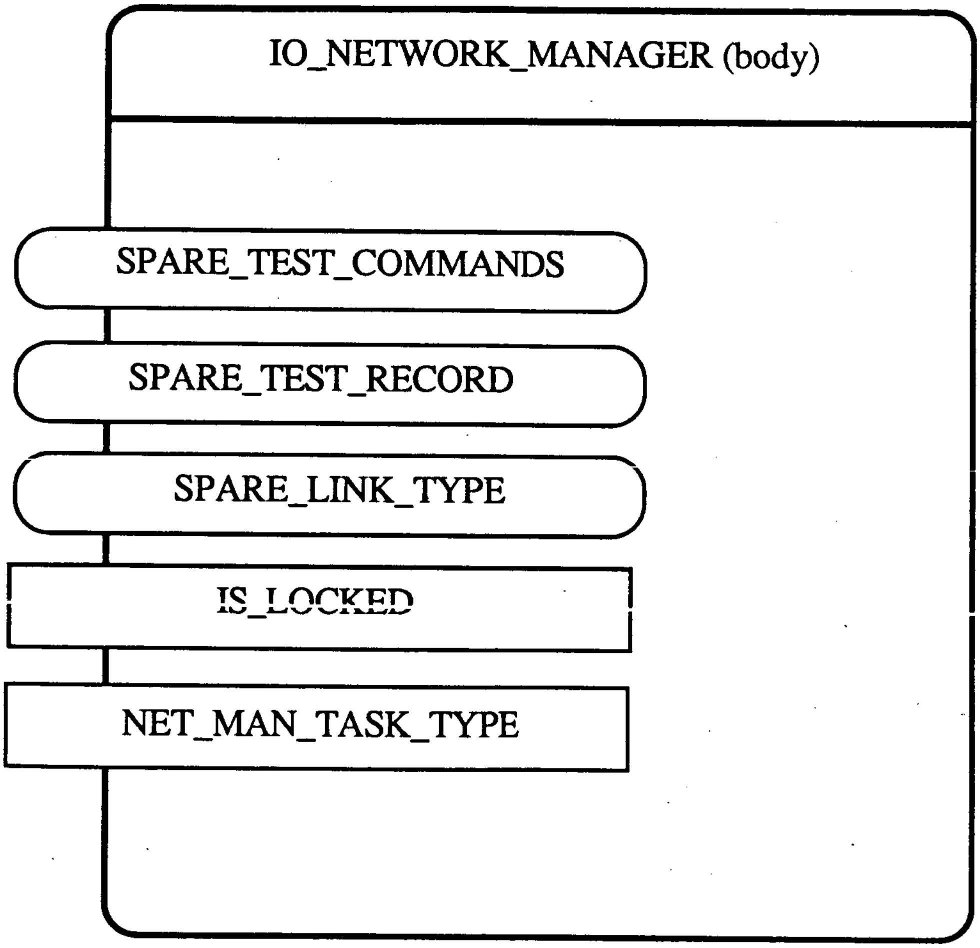


```
?? Is the channel ok ??
  no
    log error
    report failed channel
  yes
    Set up report to reflect the assumption that everything is ok
    Set up pointer to dpm and chain to run with poll
    Save branch_to address of last transaction (to next transaction)
      to be restored after chain completion
    Branch from Header to end of chain program
    Clear final csr
    Set up solicited chin pointer to run config_nodes ios program
    Wait for chain to complete and read data and restore branch
    ?? Failed a dx pattern test ??
      yes
        ?? Is the root link channel bad ??
          no
            log error to non re-alignable area and loop forever
          yes
            log error report failed channel
      no
        ?? Channel failed during chain execution
           and has been resynched ??
          yes
            log error report failed channel
          no
            ?? Is the chain complete ??
              no
                ?? stuck on high ??
                  no
                  yes
                    log error and report babbler detected
              yes
                Stop this ios in case it's babbling
                Log the error
                ?? babbler on the network ??
                  yes
                    log error and report babbler detected
                  no
                    log error and report failed ios
```

Software Specification Reference Number: 3.4.2.4



Software Specification Reference Number: 3.1



Software Specification Reference Number: 3.1

Subprogram: IS_LOCKED
Inputs: flag
Outputs: boolean flag

Test and Set the flag	
?? Is flag already set??	
yes	no
return FALSE	return TRUE

Subprogram: STOP_IO_NETWORK_MANAGER
Inputs: i/o network identifier
Outputs: stop report

?? Is net connected to this gpc ??		
no	yes	
Set stop report to net not connected to this gpc	?? Is net manager started ??	
	yes	no
	Call the stop entry in the manager of this network	Set stop report to net manger not running
	Set mgr_running to FALSE	log the event
log the event	Set stop report.had errors to FALSE	

Subprogram: STOP_IO_NETWORK_MANAGER
Inputs: i/o network identifier
Outputs: stop report

?? Is net connected to this gpc ??		
no	yes	
Set stop report to net not connected to this gpc	?? Is net manager started ??	
	yes	no
	Call the stop entry in the manager of this network	Set stop report to net manger not running
	Set mgr_running to FALSE	log the event
log the event	Set stop report.had errors to FALSE	

Subprogram: REPAIR_NETWORK
Inputs: i/o network identifier
Outputs: manager accepted call

Call repair entry in the manager of this network	
?? Entry accepted ??	
yes	no
Set mgr accepted call to TRUE	Set mgr accepted call to FALSE

Software Specification Reference Number: 3.1

Subprogram: RESTORE_FAILED_NODE_OR_FAILED_IOS
Inputs: i/o network identifier
restore information
Outputs: manager accepted call

Call restore failed part entry in the manager of this network	
?? Entry accepted ??	
yes	no
Set mgr accepted call to TRUE	Set mgr accepted call to FALSE

Subprogram: GET_NETWORK_STATE
Inputs: i/o network identifier
Outputs: network state

Return the network state for this network

Subprogram: PUT_NETWORK_STATE
Inputs: i/o network identifier
new state
Outputs: network state

Assign the new state to the network state for the this network

Subprogram: GET_ACTIVE_ROOT_LINK
Inputs: i/o network identifier
Outputs: channel record of active root link

Return the active root link for this network

Subprogram: PUT_ACTIVE_ROOT_LINK
Inputs: i/o network identifier
new active root link
Outputs: active root link

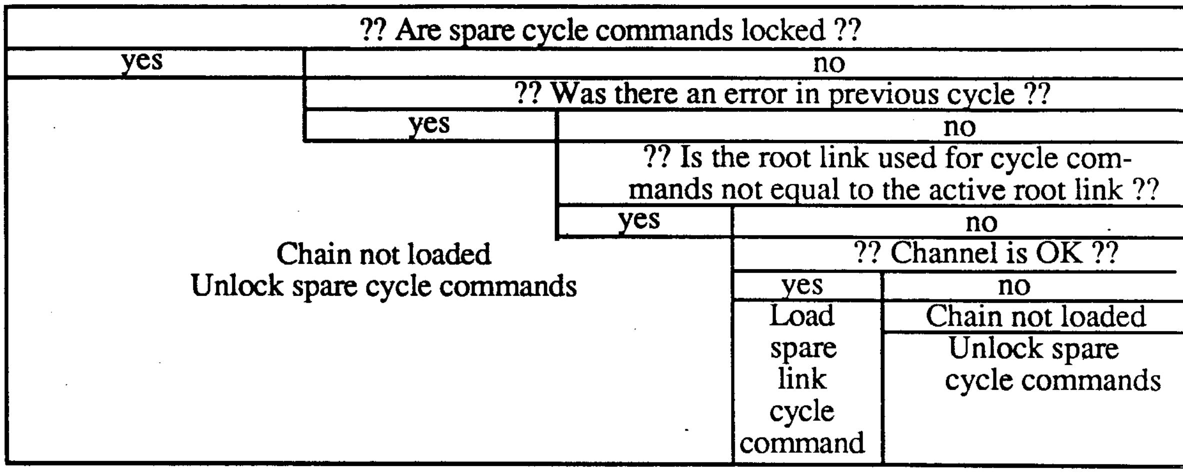
Assign the new active root link to the active root link for this network

Subprogram: EXECUTE_SPARE_LINK_TEST
Inputs: i/o network identifier
Outputs: none

Set up pointer to dpm of this network
Set up solicited chain pointer to run config nodes ios program
Start chain execution
Wait for chain to complete

Software Specification Reference Number: 3.1.2.4 , 3.1.2.5

Subprogram: LOAD SPARE LINK TEST
Inputs: I/O network identifier
Outputs chain loaded



?? Are spare cycle commands locked ??

yes

Chain not loaded
Unlock spare cycle commands

no

?? Was there an error in previous cycle ??

yes

no

?? Is the root link used for cycle commands not equal to the active root link ??

yes

no

?? Channel is OK ??

yes

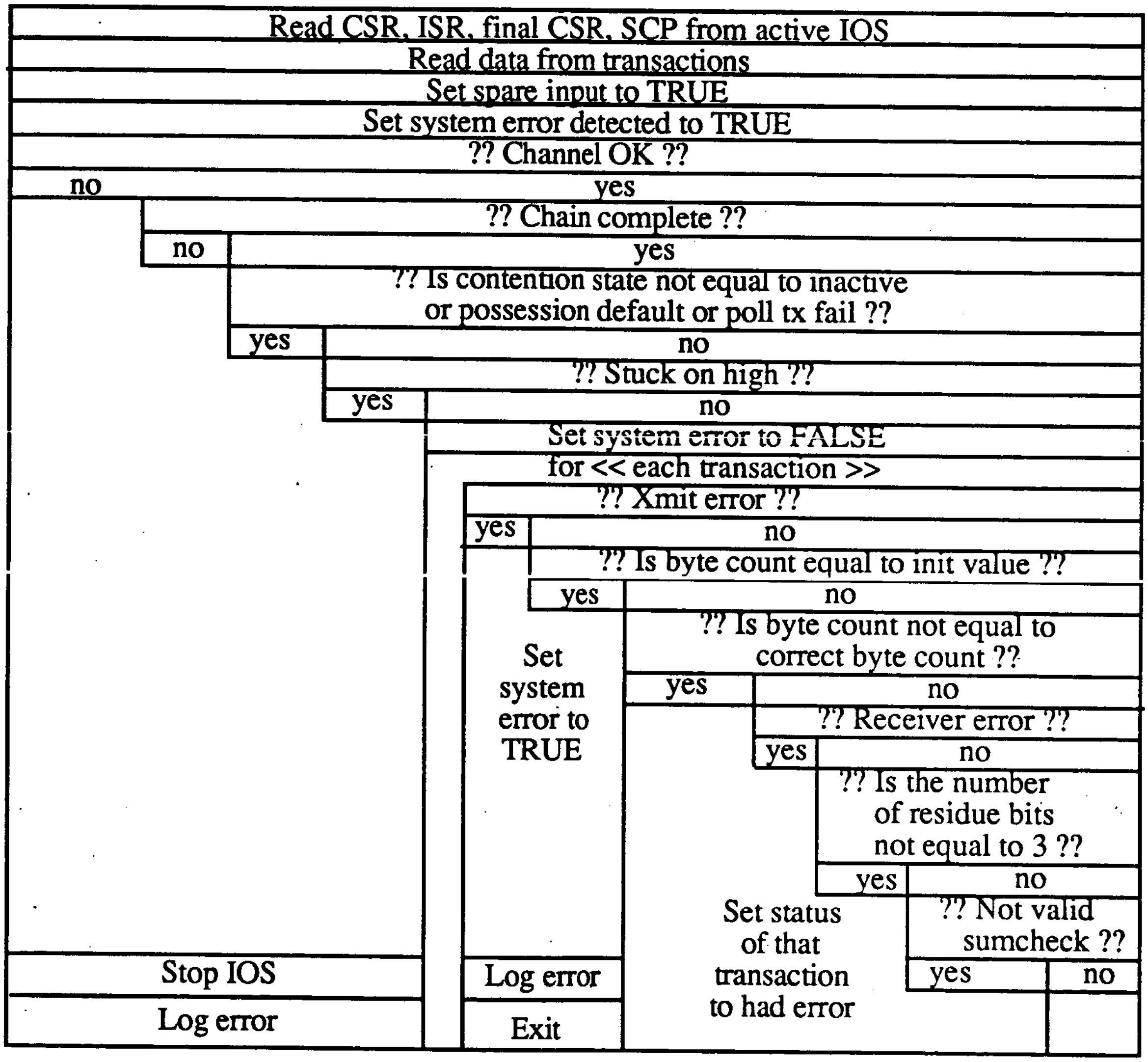
Load spare link cycle command

no

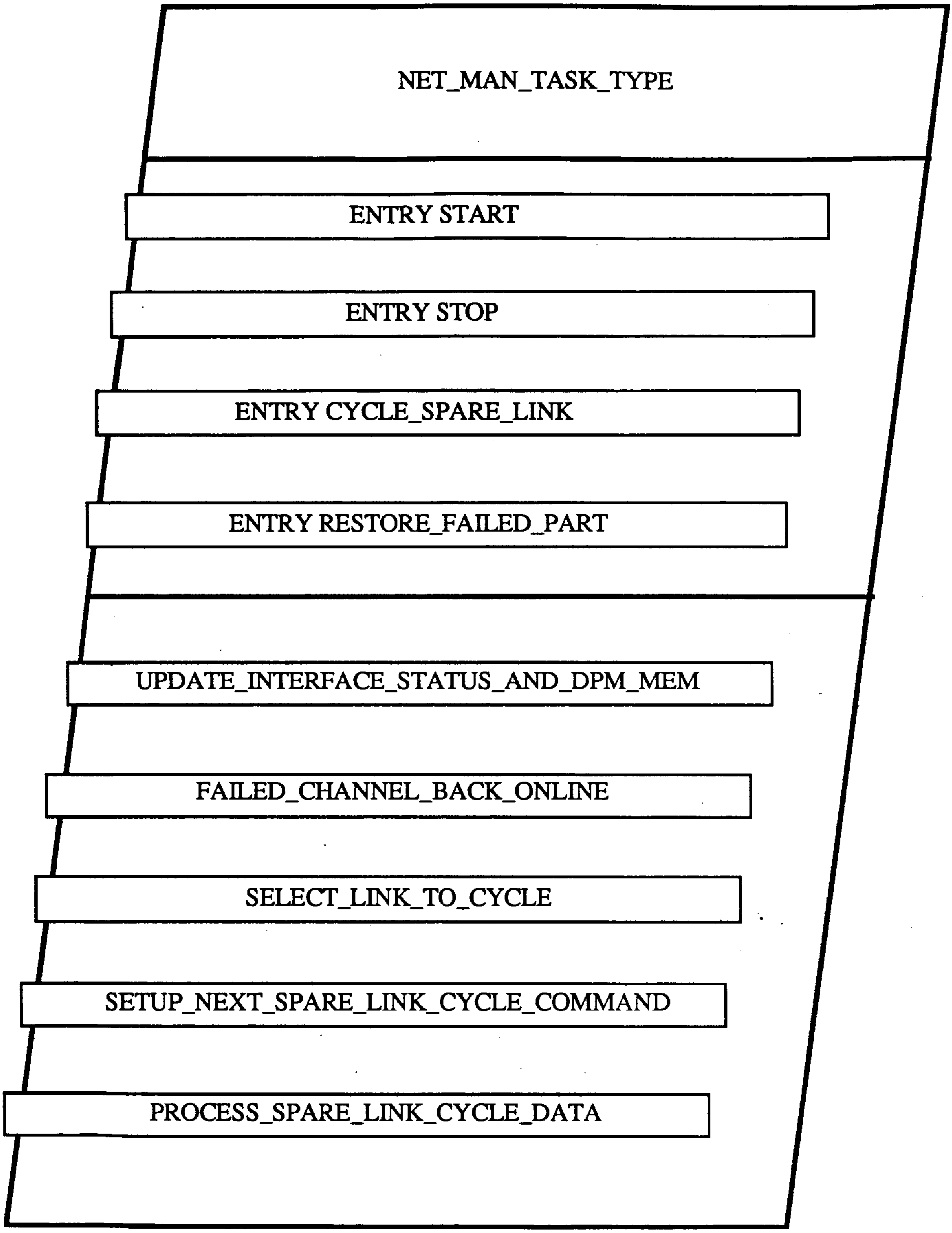
Chain not loaded
Unlock spare cycle commands

Software Specification Reference Number: 3.1.2.4

Subprogram: UPLOAD SPARE LINK TEST
Inputs: I/O network identifier
Outputs: Data from spare link cycle commands

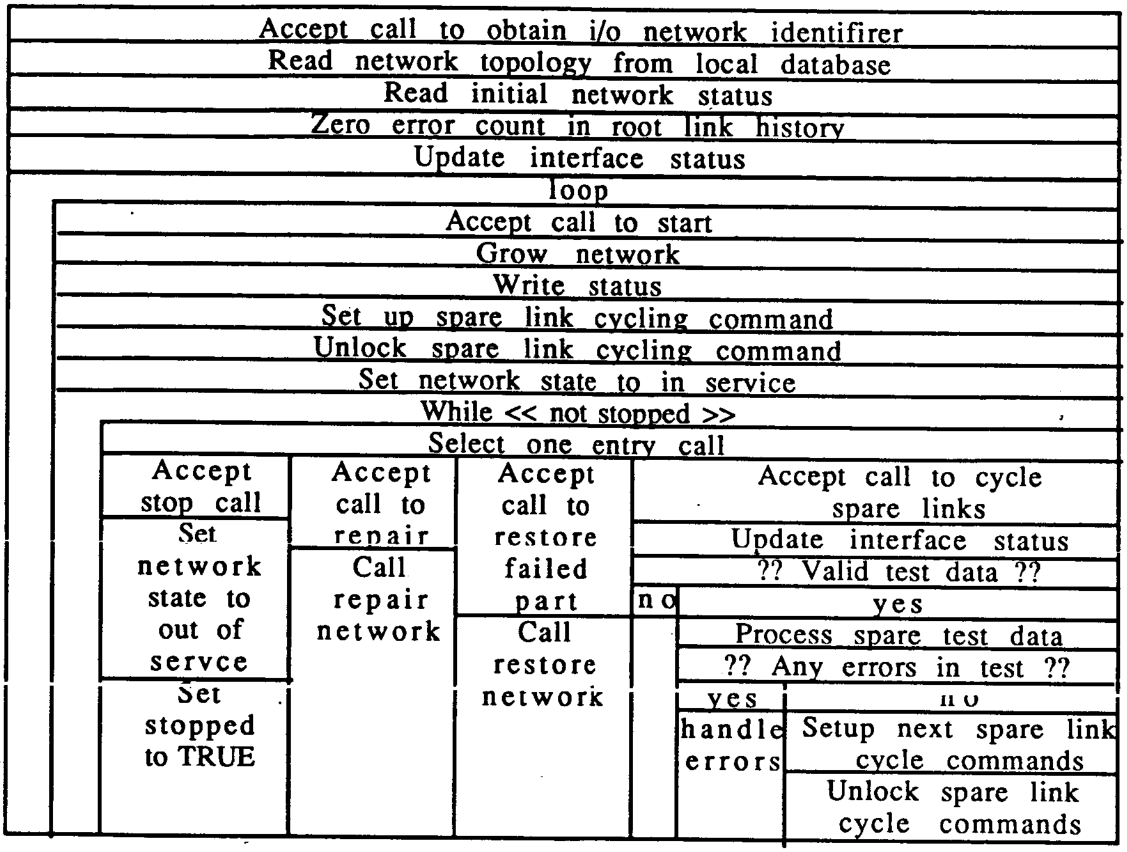

Read CSR, ISR, final CSR, SCP from active IOS
Read data from transactions
Set spare input to TRUE
Set system error detected to TRUE
?? Channel OK ??
no
yes
?? Chain complete ??
no
yes
?? Is contention state not equal to inactive or possession default or poll tx fail ??
yes
no
?? Stuck on high ??
yes
no
Set system error to FALSE
for << each transaction >>
?? Xmit error ??
yes
no
?? Is byte count equal to init value ??
yes
no
?? Is byte count not equal to correct byte count ??
yes
no
?? Receiver error ??
yes
no
?? Is the number of residue bits not equal to 3 ??
yes
no
?? Not valid sumcheck ??
yes
no
Set system error to TRUE
Set status of that transaction to had error
Stop IOS
Log error
Log error
Exit

Software Specification Reference Number: 3.1.2.4


NET_MAN_TASK_TYPE
ENTRY START
ENTRY STOP
ENTRY CYCLE_SPARE_LINK
ENTRY RESTORE_FAILED_PART
UPDATE_INTERFACE_STATUS_AND_DPM_MEM
FAILED_CHANNEL_BACK_ONLINE
SELECT_LINK_TO_CYCLE
SETUP_NEXT_SPARE_LINK_CYCLE_COMMAND
PROCESS_SPARE_LINK_CYCLE_DATA

Software Specification Reference Number: 3.1

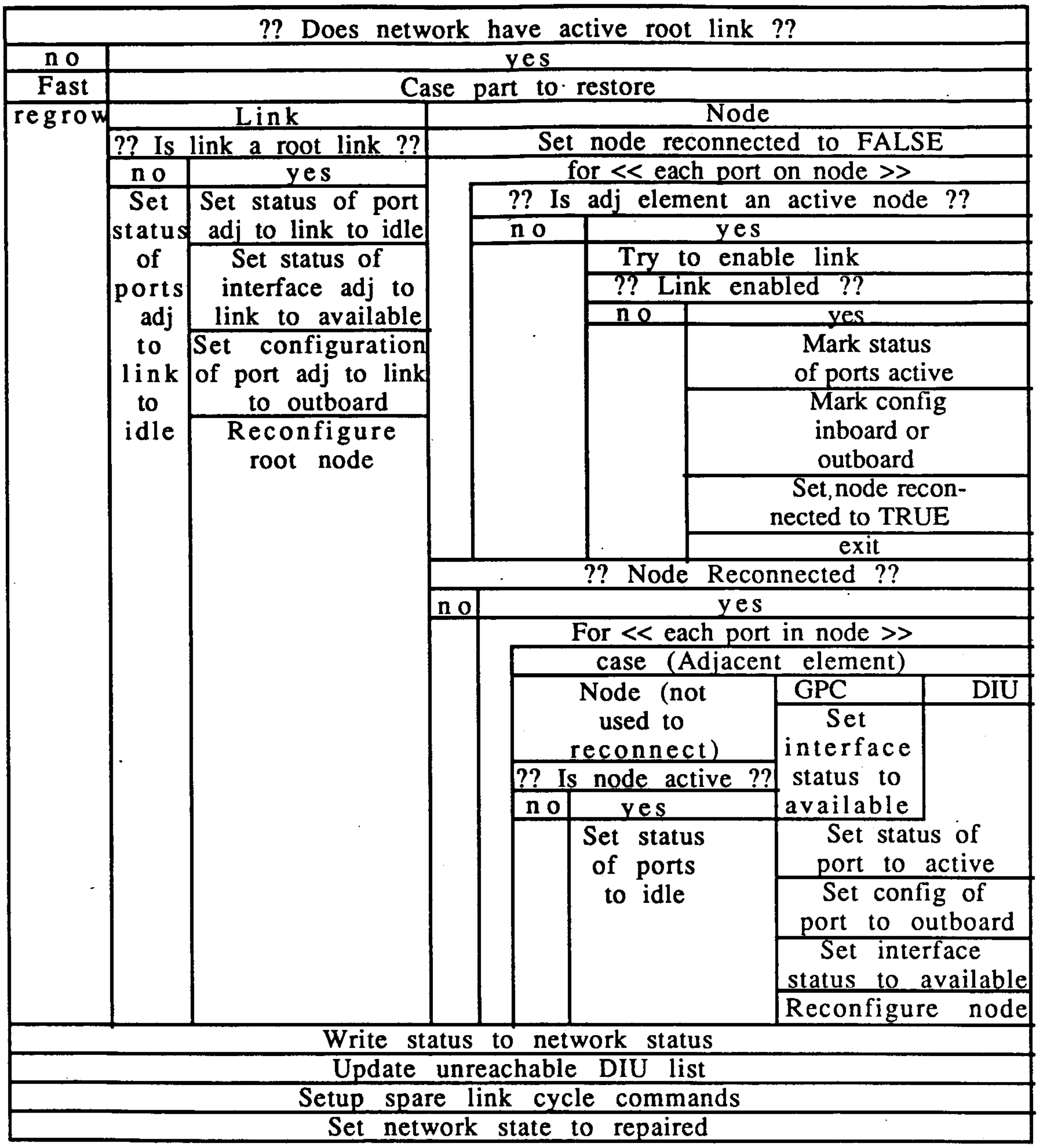
Subprogram: NET_MAN_TASK_TYPE
Inputs: i/o network identifier
network topology
network status
Outputs: network status
network state



```
Accept call to obtain i/o network identifirer
Read network topology from local database
Read initial network status
Zero error count in root link history
Update interface status
loop
    Accept call to start
    Grow network
    Write status
    Set up spare link cycling command
    Unlock spare link cycling command
    Set network state to in service
    While << not stopped >>
        Select one entry call
            Accept stop call
                Set network state to out of servce
                Set stopped to TRUE
            Accept call to repair
                Call repair network
            Accept call to restore failed part
                Call restore network
            Accept call to cycle spare links
                Update interface status
                ?? Valid test data ??
                    no
                    yes
                        Process spare test data
                        ?? Any errors in test ??
                            yes
                                handle errors
                            no
                                Setup next spare link cycle commands
                                Unlock spare link cycle commands
```

Software Specification Reference Number: 3.1

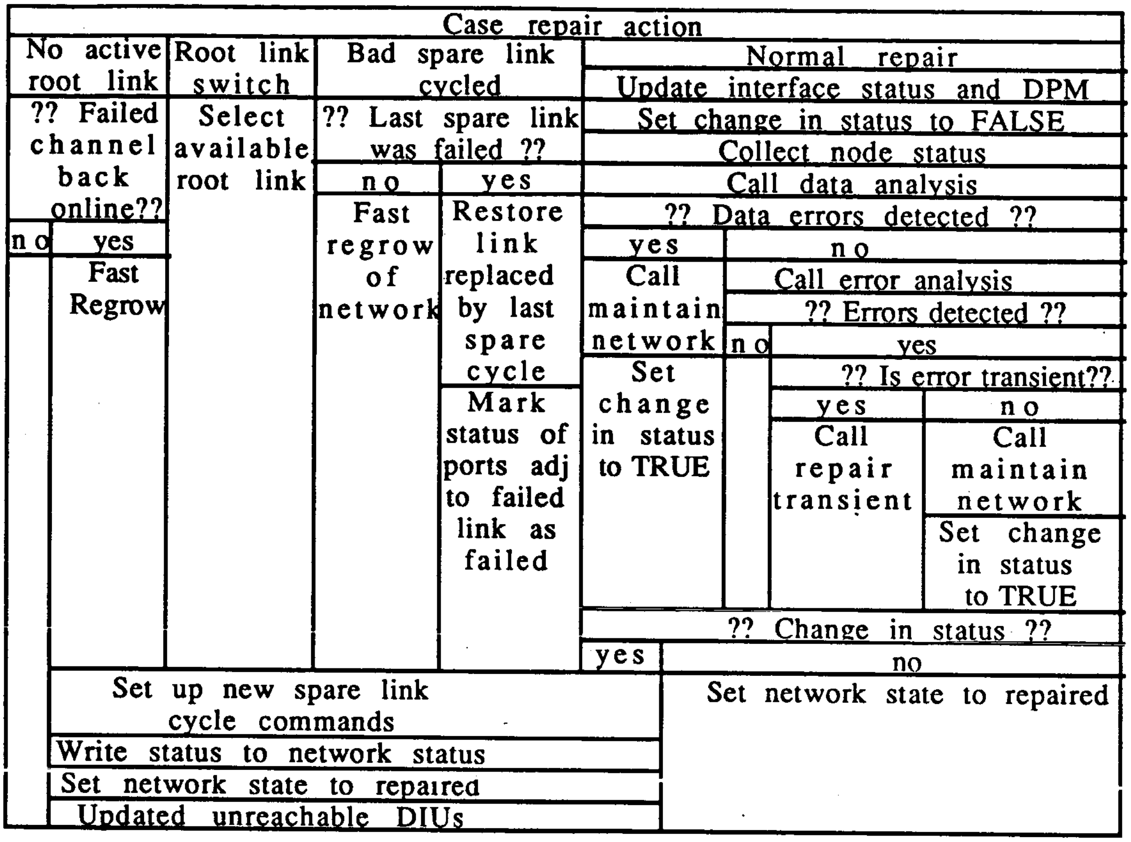
Subprogram: RESTORE_NETWORK
Inputs: part to restore
Outputs: network status



```
?? Does network have active root link ??
no
    Fast regrow
yes
    Case part to restore
    Link
        ?? Is link a root link ??
        no
            Set status of ports adj to link to idle
        yes
            Set status of port adj to link to idle
            Set status of interface adj to link to available
            Set configuration of port adj to link to outboard
            Reconfigure root node
    Node
        Set node reconnected to FALSE
        for << each port on node >>
            ?? Is adj element an active node ??
            no
            yes
                Try to enable link
                ?? Link enabled ??
                no
                yes
                    Mark status of ports active
                    Mark config inboard or outboard
                    Set,node reconnected to TRUE
                    exit
        ?? Node Reconnected ??
        no
        yes
            For << each port in node >>
                case (Adjacent element)
                Node (not used to reconnect)
                    ?? Is node active ??
                    no
                    yes
                        Set status of ports to idle
                GPC                          DIU
                    Set interface status to available
                    Set status of port to active
                    Set config of port to outboard
                    Set interface status to available
                    Reconfigure node
Write status to network status
Update unreachable DIU list
Setup spare link cycle commands
Set network state to repaired
```

Software Specification Reference Number: 3.1.2.5

Subprogram: REPAIR_NETWORK
Inputs: none
Outputs: none



```
Case repair action
    No active root link
        ?? Failed channel back online??
            no
            yes
                Fast Regrow
        Set up new spare link cycle commands
        Write status to network status
        Set network state to repaired
        Updated unreachable DIUs
    Root link switch
        Select available root link
        Set up new spare link cycle commands
        Write status to network status
        Set network state to repaired
        Updated unreachable DIUs
    Bad spare link cycled
        ?? Last spare link was failed ??
            no
                Fast regrow of network
            yes
                Restore link replaced by last spare cycle
                Mark status of ports adj to failed link as failed
        Set up new spare link cycle commands
        Write status to network status
        Set network state to repaired
        Updated unreachable DIUs
    Normal repair
        Update interface status and DPM
        Set change in status to FALSE
        Collect node status
        Call data analysis
        ?? Data errors detected ??
            yes
                Call maintain network
                Set change in status to TRUE
            no
                Call error analysis
                ?? Errors detected ??
                    no
                    yes
                        ?? Is error transient??
                            yes
                                Call repair transient
                            no
                                Call maintain network
                                Set change in status to TRUE
        ?? Change in status ??
            yes
                Set up new spare link cycle commands
                Write status to network status
                Set network state to repaired
                Updated unreachable DIUs
            no
                Set network state to repaired
```

Software Specification Reference Number: 3.1.2.3

Subprogram: UPDATE_INTERFACE_STATUS_AND_DPM_MEM
Inputs: interface status
GPC FDIR status
Outputs: network status
updated DPM

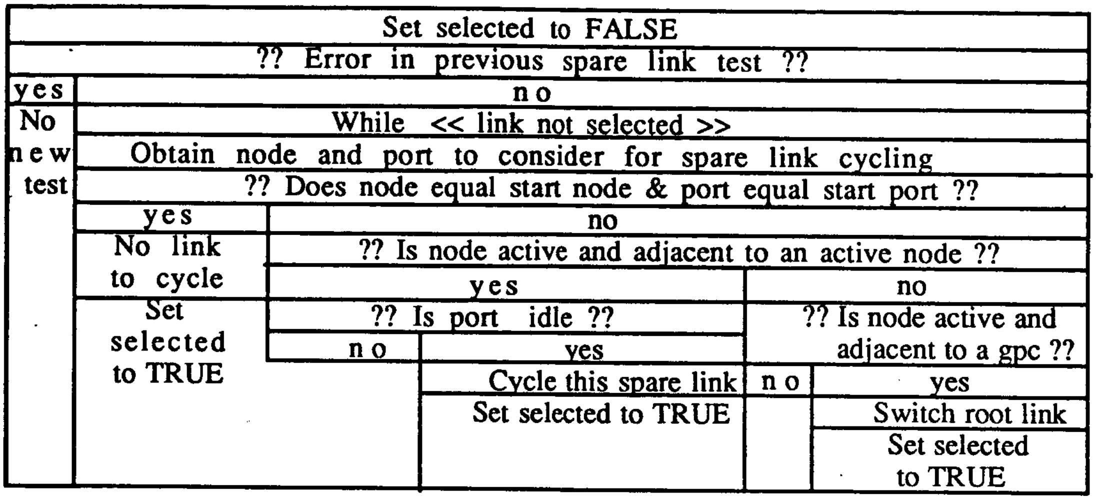
```
Set change in interface status to false
for << each interface >>
    ?? Does interface status equal failed channel ??
    yes
        ?? Is the channel ok ??
        no
        yes
            Set interface status to available
            Update status chain
            Set change in interface status to TRUE
    no
        ?? Does interface status equal available ??
        yes
            ?? Is the channel not ok ??
            yes
                Set the interface status to failed channel
                Set change in interface status to TRUE
            no
        no
?? Is there a change in the interface status ??
no
yes
    Write the network status
```

Subprogram: FAILED_CHANNEL_BACK_ONLINE
Inputs: interface status
GPC FDIR status
Outputs: boolean flag

```
for << each interface >>
    ?? Does the interface status equal failed channel and
       the channel is now ok ??
    no
    yes
        Log the event the channel is back online
        Return TRUE
Return FALSE
```

Software Specification Reference Number: 3.1.2 , 3.1.2.4

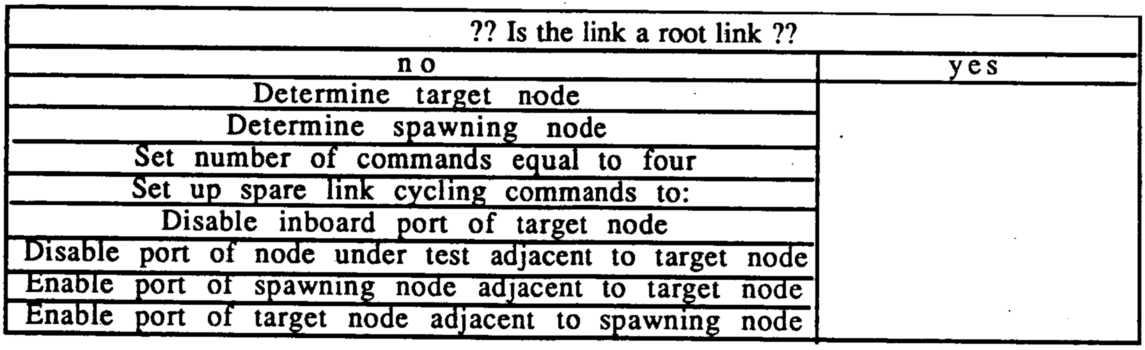
Subprogram: SELECT_LINK_TO_CYCLE
Inputs: network topology
network status
Outputs: spare link to test



```
Set selected to FALSE
?? Error in previous spare link test ??
  yes
    No new test
  no
    While << link not selected >>
      Obtain node and port to consider for spare link cycling
      ?? Does node equal start node & port equal start port ??
        yes
          No link to cycle
          Set selected to TRUE
        no
          ?? Is node active and adjacent to an active node ??
            yes
              ?? Is port idle ??
                no
                yes
                  Cycle this spare link
                  Set selected to TRUE
            no
              ?? Is node active and adjacent to a gpc ??
                no
                yes
                  Switch root link
                  Set selected to TRUE
```

Software Specification Reference Number: 3.1.2 , 3.1.2.4

Subprogram: SET_UP_NEXT_SPARE_LINK_CYCLE_COMMANDS
Inputs: spare to cycle
Outputs: spare link cycle commands

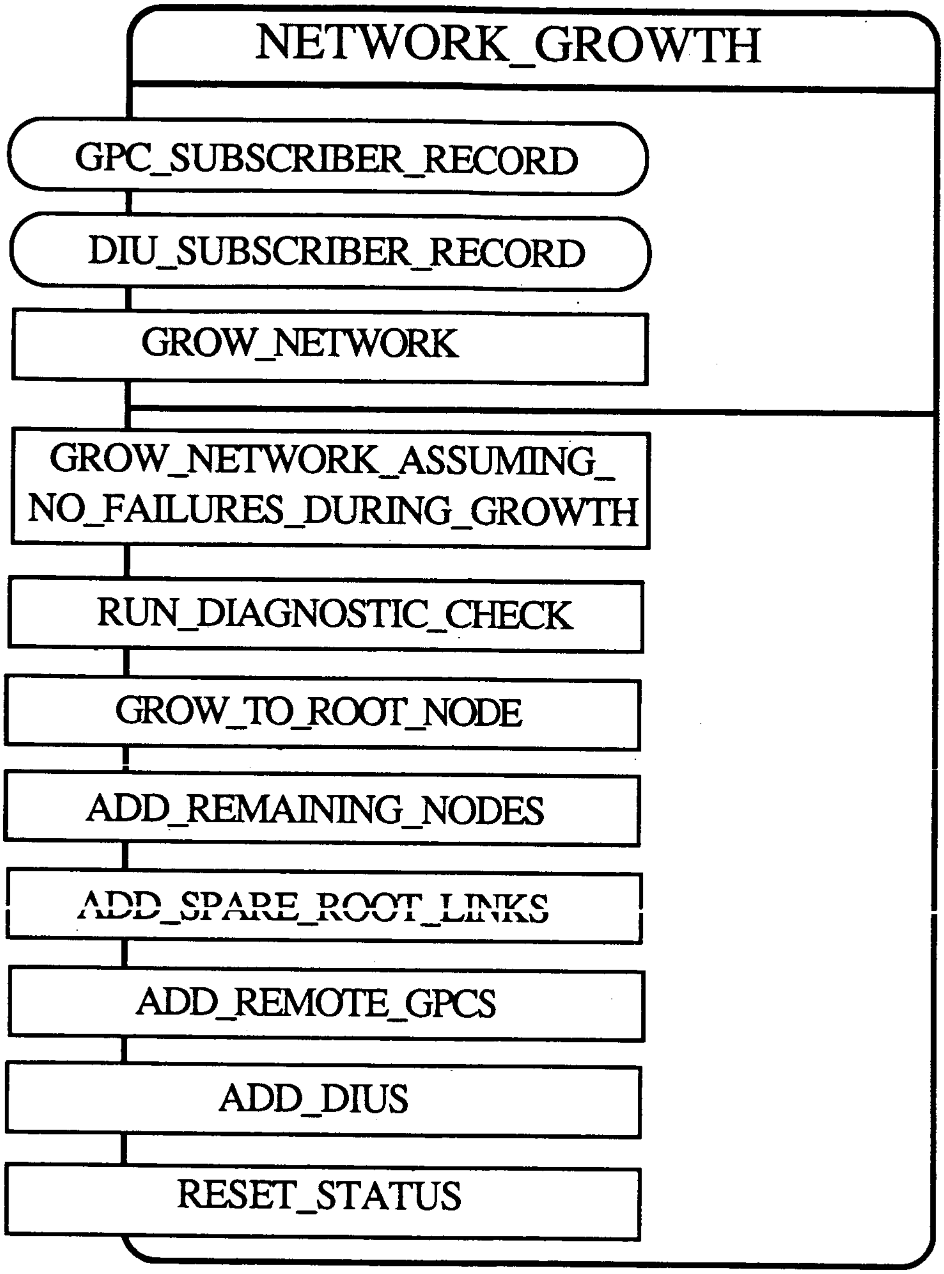


```
?? Is the link a root link ??
  no
    Determine target node
    Determine spawning node
    Set number of commands equal to four
    Set up spare link cycling commands to:
    Disable inboard port of target node
    Disable port of node under test adjacent to target node
    Enable port of spawning node adjacent to target node
    Enable port of target node adjacent to spawning node
  yes
```

Subprogram: PROCESS_SPARE_LINK_CYCLE_DATA
Inputs: responses to spare cycle commands
Outputs: error flags

<table>
<tr><td colspan="6">?? Is the link a root link ??</td></tr>
<tr><td>yes</td><td colspan="5">no</td></tr>
<tr><td rowspan="9"></td><td colspan="5">?? System error detected ??</td></tr>
<tr><td>yes</td><td colspan="4">no</td></tr>
<tr><td rowspan="7">Set regrow flag</td><td colspan="4">?? Normal response pattern ??</td></tr>
<tr><td>yes</td><td colspan="3">no</td></tr>
<tr><td rowspan="5">Update status and configuration</td><td colspan="3">?? Is it a bad link ??</td></tr>
<tr><td>yes</td><td colspan="2">no</td></tr>
<tr><td rowspan="3">Set recover spare link test flag, and save port/node to be recovered</td><td colspan="2">?? Is there no response from all transactions or response from all ??</td></tr>
<tr><td>yes</td><td>no</td></tr>
<tr><td>Set spare link error flag to call maintain on next repair call</td><td>Set regrow flag</td></tr>
</table>

Software Specification Reference Number: 3.1.2.4



Software Specification Reference Number: 3.1.1

Subprogram: GROW_NETWORK

Inputs:
I/O Network Interface
Network Topology
Root Links
Node Configuration
Network Status
Current Channel
Fast Grow Flag

Outputs:
Node Configuration
Network Status
Current Channel

<table>
<tr><td colspan="3">Grow_Network_Assuming_No_Failures_During_Growth</td></tr>
<tr><td colspan="3">Monitor_Network</td></tr>
<tr><td colspan="3">?? Did monitor chain find faults ??</td></tr>
<tr><td>no</td><td colspan="2">yes</td></tr>
<tr><td rowspan="3"></td><td colspan="2">?? Has growth reached max tries ??</td></tr>
<tr><td>no</td><td>yes</td></tr>
<tr><td>Loop Back
to
Grow_Network_Assuming_No_Failures_During_Growth</td><td>Set node status for all nodes to failed</td></tr>
</table>

Subprogram: RESET_STATUS

Inputs:
Network Status
Node Configuration

Outputs:
Network Status
Node Configuration

Set status of all nodes to idle
Set status of all ports to idle
Set all ports in all nodes in node configuration to idleport
Network is not active

Software Specification Reference Number: 3.1.1

Subprogram: GROW_NETWORK_ASSUMING_NO_FAILURE_DURING_GROWTH

Inputs:
i/o network identifier
network topology
root links
node configuration
network status
current channel
grow successful flag
fast grow flag

Outputs:
node configuation
network status
current channel
growth successful flag

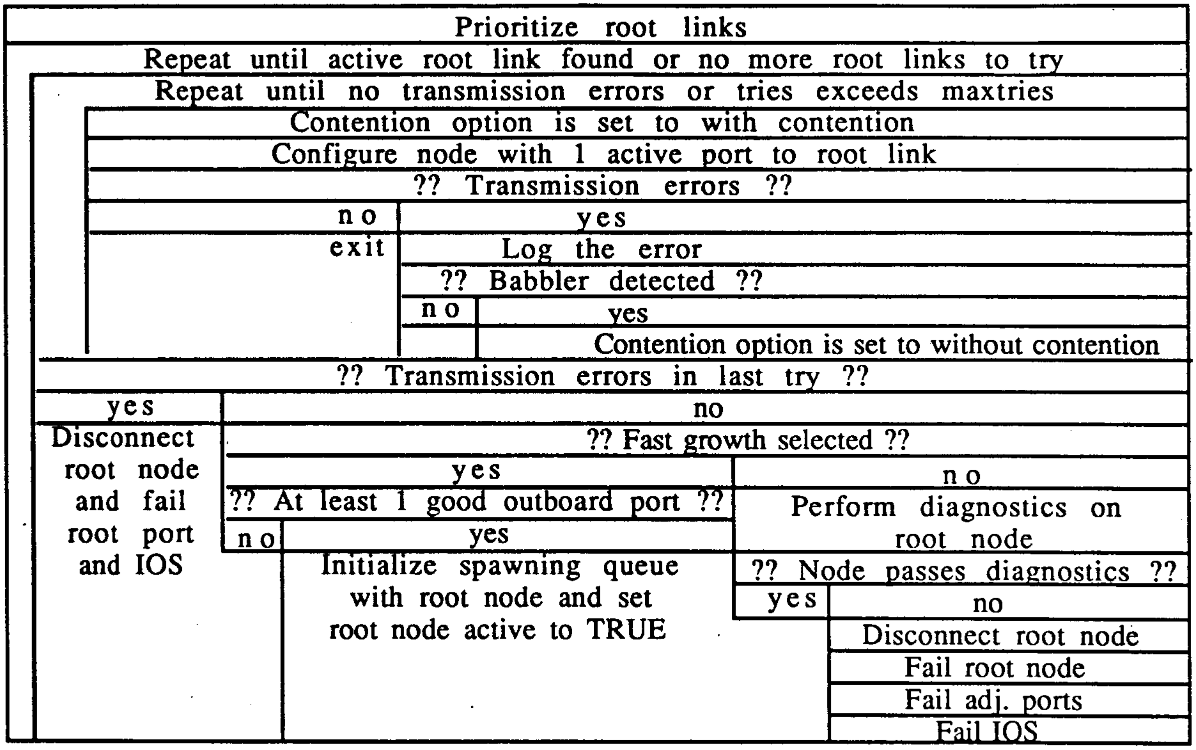
```
Reset status.
Set grow successful to FALSE.
Grow to root node.
?? Is root node active ??
    no
    yes
        Add remaining nodes
        Set status of nonactive nodes to failed
        Add spare root links
        Add dius
        Add remote gpcs
        Set Grow successful to TRUE
```

Software Specification Reference Number: 3.1.1

Subprogram: GROW_TO_ROOT_NODE

Inputs: i/o network identifier
root links
network topology
network status
node configuration
spawning queue
next entry
root node active flag
fast grow flag

Outputs: network status
node configuration
spawning queue
next entry
root node active flag
fast grow flag

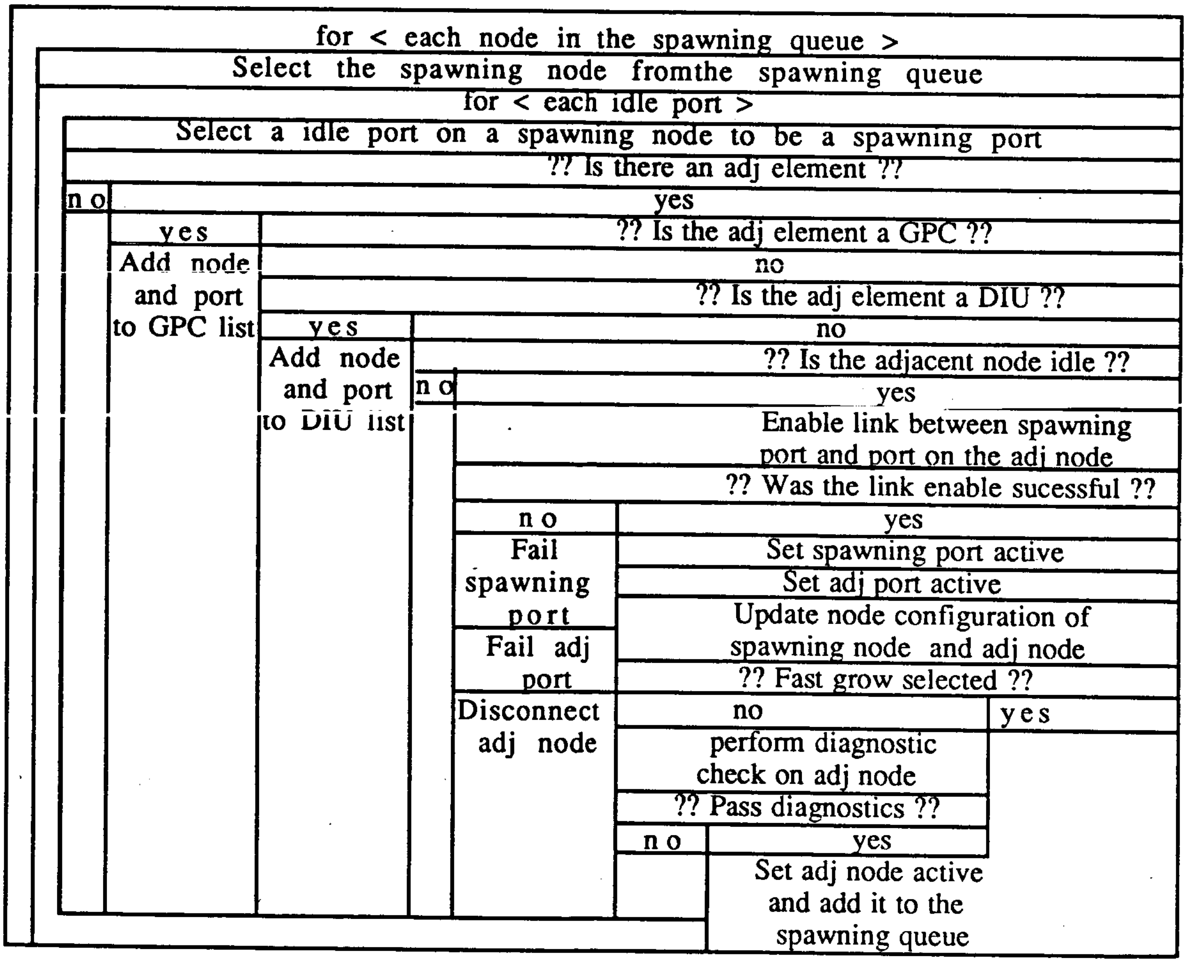


```
Prioritize root links
Repeat until active root link found or no more root links to try
    Repeat until no transmission errors or tries exceeds maxtries
        Contention option is set to with contention
        Configure node with 1 active port to root link
        ?? Transmission errors ??
            no
                exit
            yes
                Log the error
                ?? Babbler detected ??
                    no
                    yes
                        Contention option is set to without contention
    ?? Transmission errors in last try ??
        yes
            Disconnect root node and fail root port and IOS
        no
            ?? Fast growth selected ??
                yes
                    ?? At least 1 good outboard port ??
                        no
                        yes
                            Initialize spawning queue with root node and set root node active to TRUE
                no
                    Perform diagnostics on root node
                    ?? Node passes diagnostics ??
                        yes
                        no
                            Disconnect root node
                            Fail root node
                            Fail adj. ports
                            Fail IOS
```

Software Specification Reference Number: 3.1.1.1

Subprogram: ADD_REMAINING_NODES

Inputs:	i/o network identifier
	root links
	network topology
	network status
	node configuration
	spawning queue
	next entry
	network subscribers
	fast grow flag
Outputs:	network status
	node configuration
	spawning queue
	next entry
	network subscribers
	fast grow flag

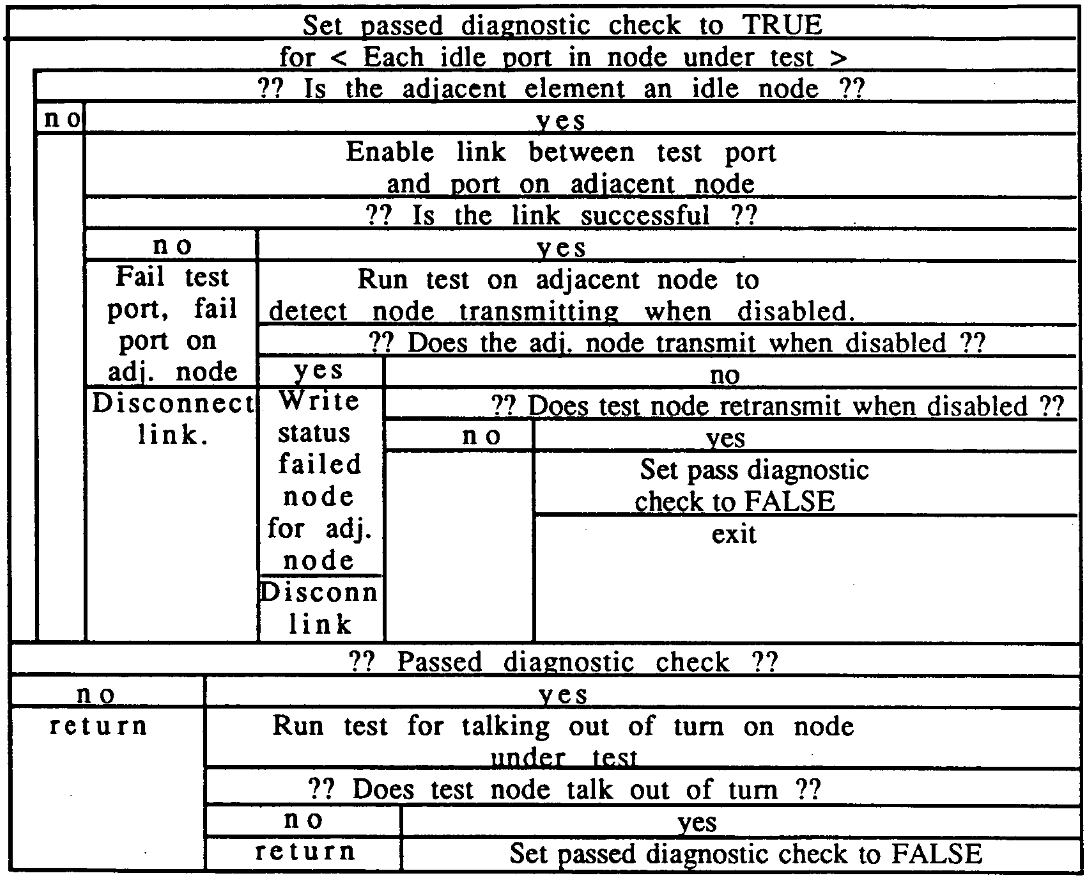


```
for < each node in the spawning queue >
    Select the spawning node fromthe spawning queue
    for < each idle port >
        Select a idle port on a spawning node to be a spawning port
        ?? Is there an adj element ??
        yes
            ?? Is the adj element a GPC ??
            yes
                Add node and port to GPC list
            no
                ?? Is the adj element a DIU ??
                yes
                    Add node and port to DIU list
                no
                    ?? Is the adjacent node idle ??
                    yes
                        Enable link between spawning port and port on the adj node
                        ?? Was the link enable sucessful ??
                        yes
                            Set spawning port active
                            Set adj port active
                            Update node configuration of spawning node and adj node
                            ?? Fast grow selected ??
                            no
                                perform diagnostic check on adj node
                                ?? Pass diagnostics ??
                                yes
                                    Set adj node active and add it to the spawning queue
                                no
                            yes
                        no
                            Fail spawning port
                            Fail adj port
                            Disconnect adj node
                    no
        no
```

Software Specification Reference Number: 3.1.1.2

Subprogram: RUN_DIAGNOSTIC CHECK

Inputs:	node under test i/o network identifier network topology inboard port of node under test network configuration current channel network status passed diagnostic test
Outputs:	network configuration network status passed diagnostic test



```
Set passed diagnostic check to TRUE
for < Each idle port in node under test >
    ?? Is the adjacent element an idle node ??
    no
    yes
        Enable link between test port
        and port on adjacent node
        ?? Is the link successful ??
        no
            Fail test port, fail port on adj. node
            Disconnect link.
        yes
            Run test on adjacent node to
            detect node transmitting when disabled.
            ?? Does the adj. node transmit when disabled ??
            yes
                Write status failed node for adj. node
                Disconn link
            no
                ?? Does test node retransmit when disabled ??
                no
                yes
                    Set pass diagnostic check to FALSE
                    exit
?? Passed diagnostic check ??
no
    return
yes
    Run test for talking out of turn on node under test
    ?? Does test node talk out of turn ??
    no
        return
    yes
        Set passed diagnostic check to FALSE
```

Software Specification Reference Number: 3.1.1.3

Subprogram: NODE_TRANSMITS_WHEN_DISABLED
Inputs: node under test
port under test
Outputs: boolean flag

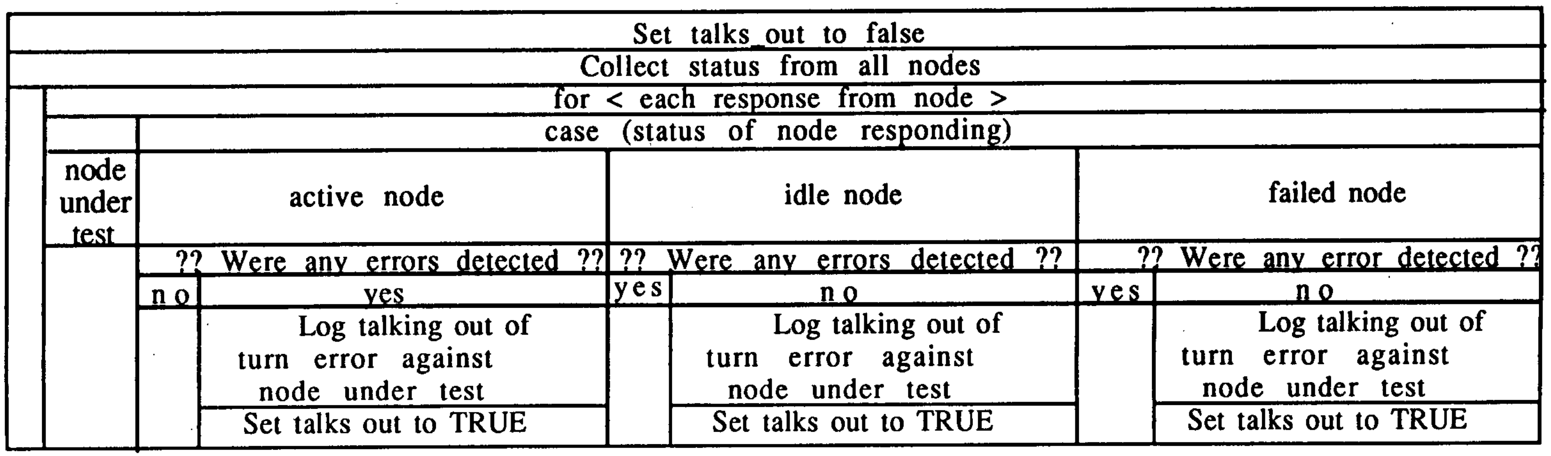
Configure node under test to disable all ports	
?? Does node respond ??	
no	yes
return FALSE	Log transmit when disabled error against this node
	return TRUE

Subprogram: NODE_RETRANSMITS_WHEN_DISABLED
Inputs: node under test
port under test
inboard port
adjacent node
one shot port
Outputs: boolean flag

Configure node under test with only inboard port enabled	
Configure one shot port on adjacent node for one transmission only	
Command node under test to return status	
?? Response from adjacent node or valid fram seen from adj. node ??	
no	yes
return FALSE	Log retansmit when disabled error against this node
	return TRUE

Software Specification Reference Number: 3.1.1.3

Subprogram: NODE_TALKS_OUT_OF_TURN
Inputs: node under test
inboard port
Outputs: boolean flag



```
Set talks_out to false
Collect status from all nodes
for < each response from node >
    case (status of node responding)
        node under test
        active node
            ?? Were any errors detected ??
                no
                yes
                    Log talking out of turn error against node under test
                    Set talks out to TRUE
        idle node
            ?? Were any errors detected ??
                yes
                no
                    Log talking out of turn error against node under test
                    Set talks out to TRUE
        failed node
            ?? Were any error detected ??
                yes
                no
                    Log talking out of turn error against node under test
                    Set talks out to TRUE
```

Software Specification Reference Number: 3.1.1.3

Subprogram: ADD_SPARE_ROOT_LINKS

Inputs:
current channel
i/o network identifier
root links
network topology
network status
node configuration
fast grow flag

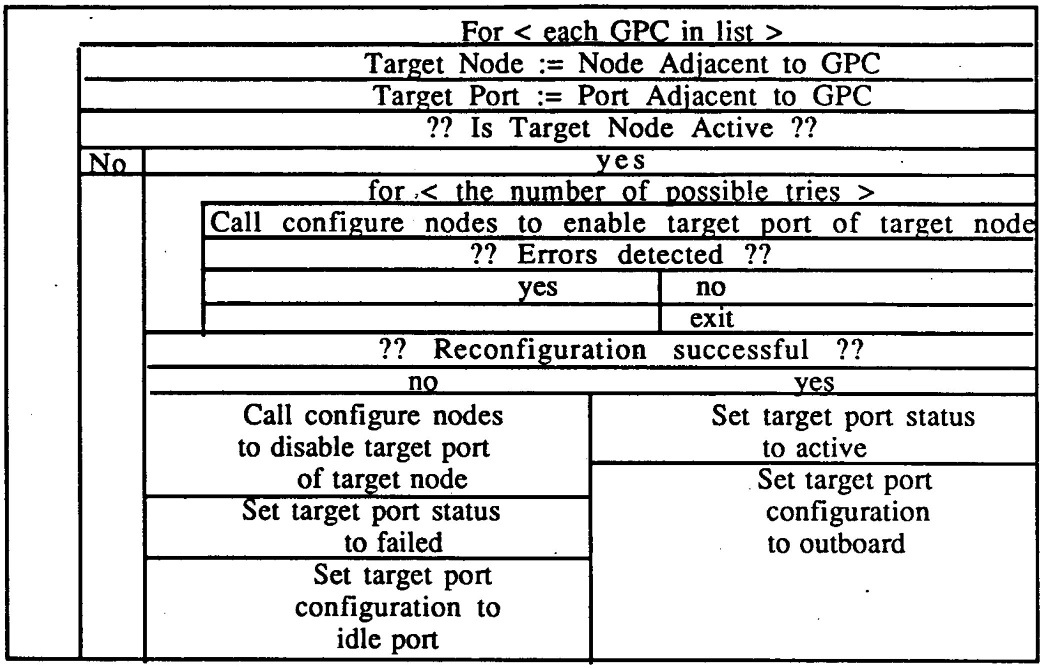
Outputs:
network status
node configuration

```
for < each root link in the network >
    ?? Is the interface status equal to idle ??
    yes
        ?? Is the root node active ??
        yes
            Repeat until root node connected or tries exceeds max tries
                Call configure node to enable root port of root node
                ?? Have any errors been detected ??
                yes
                    Need to disconnect
                no
                    Root node connected
            ?? Need to disconnect ??
            no
                Set status of root port to active
                Set configuration of root port to outboard
                Set status of ios to available
            yes
                Call configure nodes to disable root port of root node
                Fail root port
                Mark interface status failed ios
        no
    no
for < Each root link >
    ?? Is interface status available ??
    yes
        GPC FDIR says channel is failed
        yes
            Set interface states to failed channel
        no
            ?? Fast growth selected ??
            yes
            no
                Collect status using this root link
                ?? Are there any errors detected ??
                no
                yes
                    Call configure nodes to disable root port
                    Set interface status to failed ios
                    Set configuration of root part to idle port
    no
```

Software Specification Reference Number: 3.1.1.4

Subprogram: ADD_REMOTE_GPCS

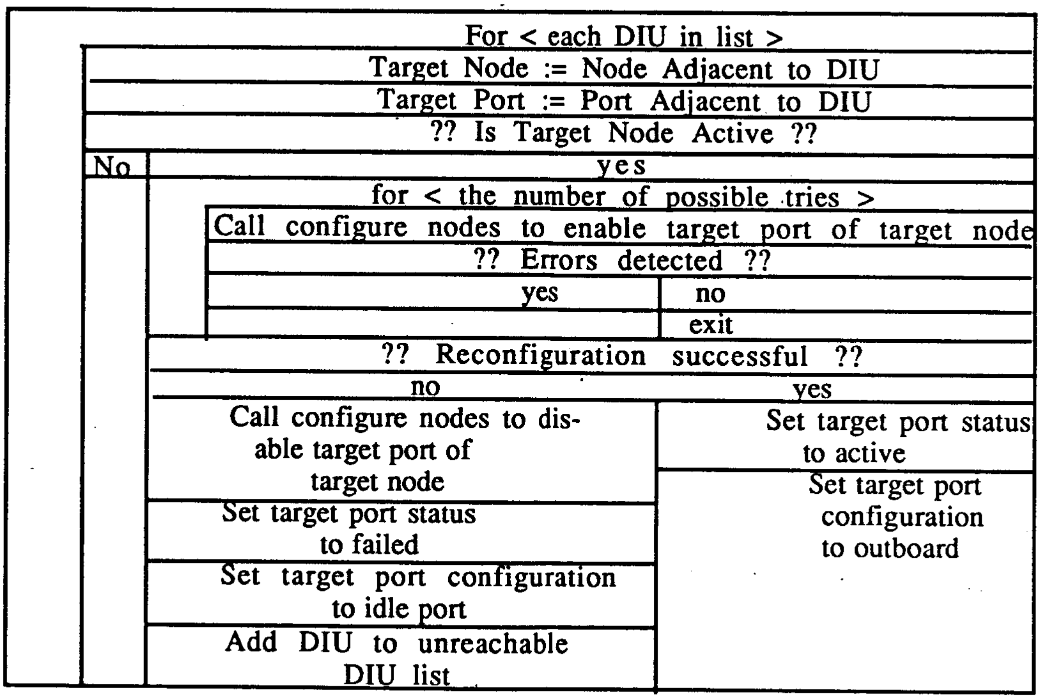
```
Inputs:   current channel
          i/o network identifier
          network topology
          GPC count
          GPC list
          network status
          network configuration
Outputs:  network status
          network configuration
```



```
For < each GPC in list >
    Target Node := Node Adjacent to GPC
    Target Port := Port Adjacent to GPC
    ?? Is Target Node Active ??
    No
    yes
        for < the number of possible tries >
            Call configure nodes to enable target port of target node
            ?? Errors detected ??
            yes
            no
                exit
        ?? Reconfiguration successful ??
        no
            Call configure nodes to disable target port of target node
            Set target port status to failed
            Set target port configuration to idle port
        yes
            Set target port status to active
            Set target port configuration to outboard
```

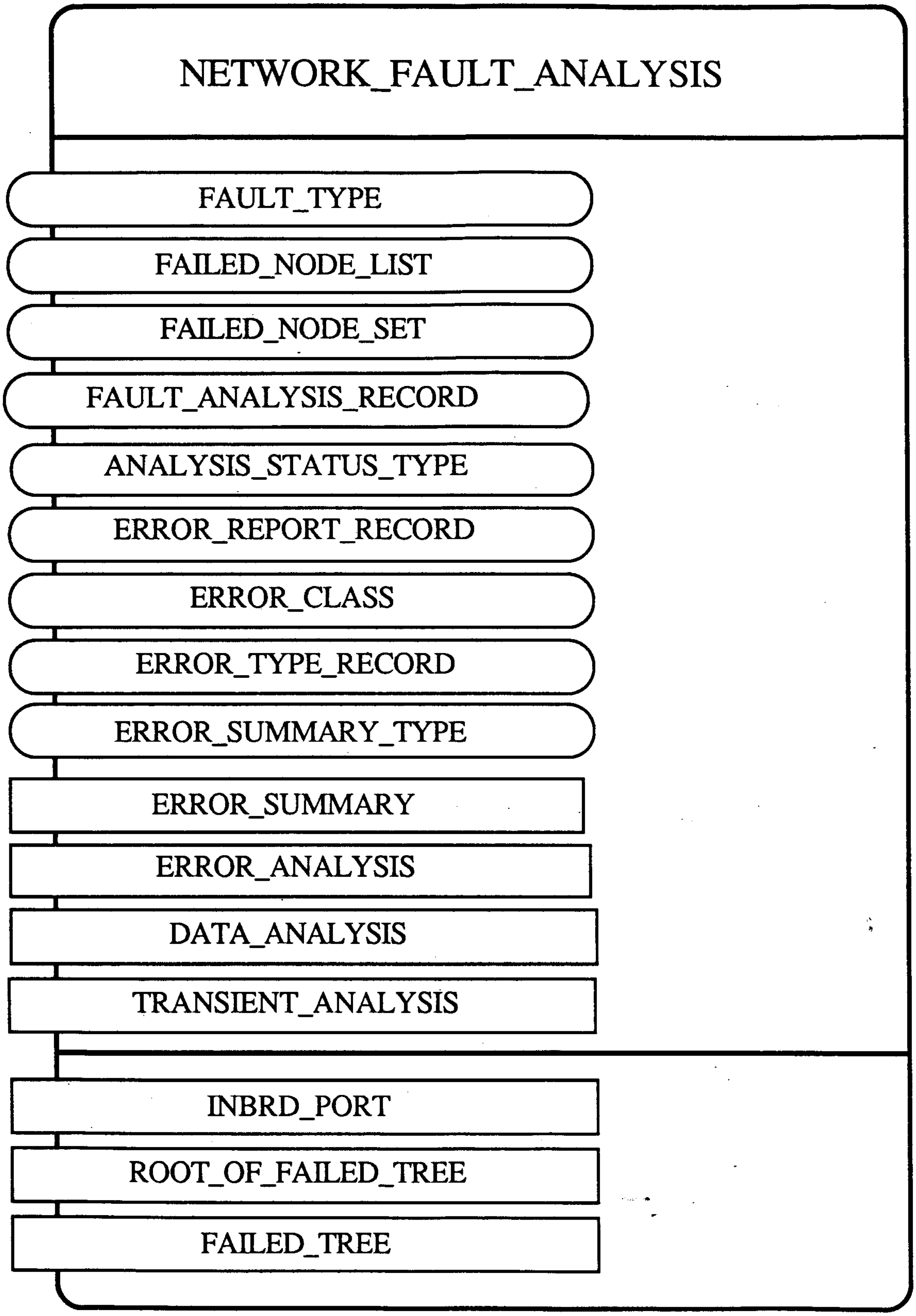
Software Specification Reference Number: 3.1.1.5

Subprogram: ADD_DIUS
Inputs: current channel
i/o network identifier
network topology
DIU count
DIU list
network status
network configuration
Outputs: network status
network configuration
unreachable DIU list



```
For < each DIU in list >
    Target Node := Node Adjacent to DIU
    Target Port := Port Adjacent to DIU
    ?? Is Target Node Active ??
    No
    yes
        for < the number of possible tries >
            Call configure nodes to enable target port of target node
            ?? Errors detected ??
            yes
            no
                exit
        ?? Reconfiguration successful ??
        no
            Call configure nodes to disable target port of target node
            Set target port status to failed
            Set target port configuration to idle port
            Add DIU to unreachable DIU list
        yes
            Set target port status to active
            Set target port configuration to outboard
```

Software Specification Reference Number: 3.1.1.6



Software Specification Reference Number: 3.1.2.2

Subprogram: ERROR_SUMMARY
Inputs: status report
Outputs: error summary

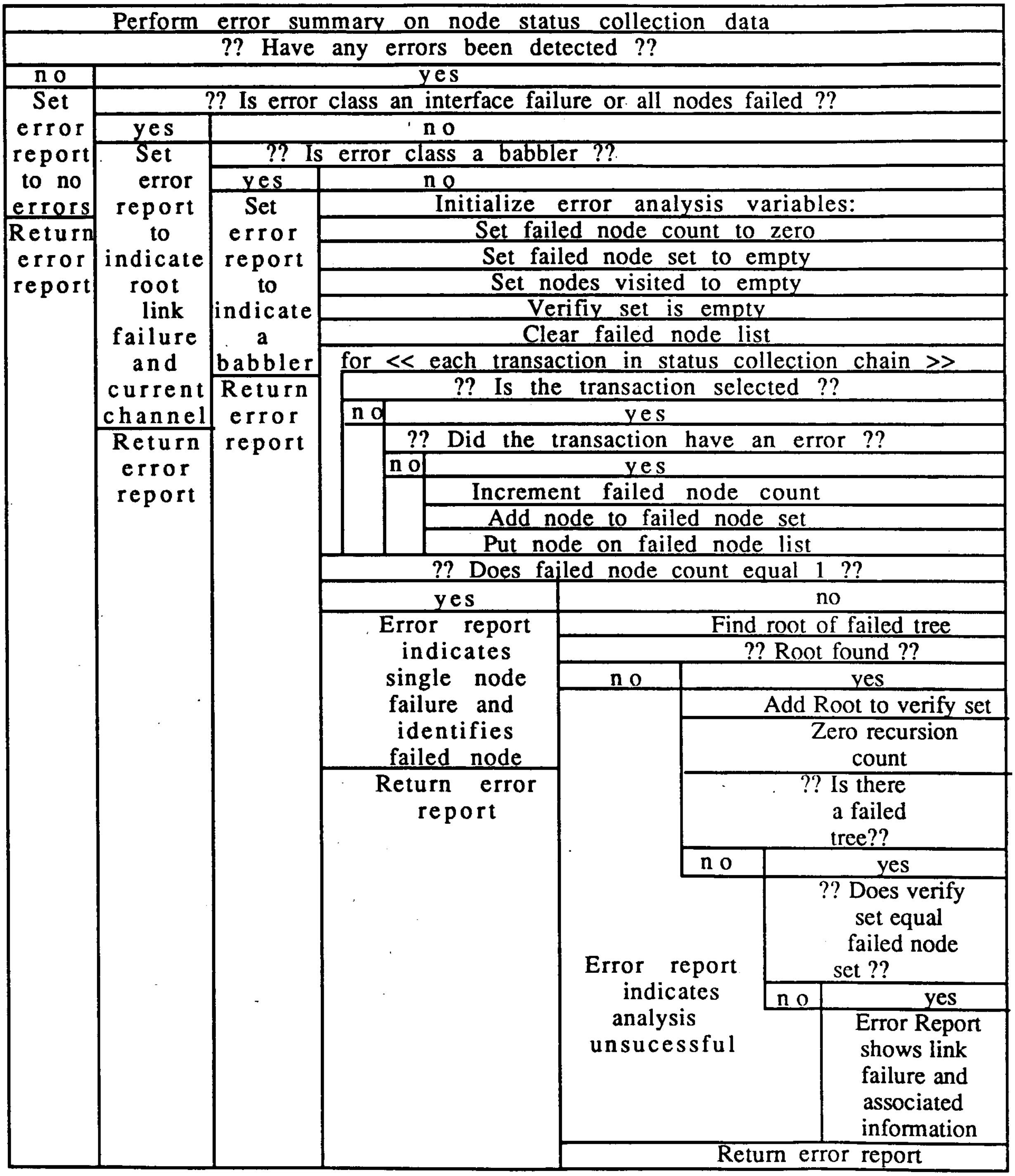
```
?? Does the status report indicate an interface failure ??
  yes
    Error summary reports any interface errors is TRUE
  no
    ?? Does the status report indicate a babbler detected ??
      yes
        Error summary reports any babbler errors is TRUE
      no
        Set any to FALSE and all to TRUE
        for << each node >>
          ?? Is the node selected ??
            no
            yes
              ?? Did the node have a error ??
                yes
                  Set any to TRUE
                no
                  Set all to FALSE
?? Is any FALSE ??
  yes
    Error summary reports any error is FALSE
  no
    ?? Is all FALSE ??
      no
        Error summary reports any error is TRUE
      yes
        Error summary reports all errors is TRUE
Return the error summary
```

Software Specification Reference Number: 3.1.2.2

Subprogram: ERROR_ANALYSIS

Inputs:
i/o network identifier
current channel
node configuration
network topology

Outputs:
error report record



```
Perform error summary on node status collection data
?? Have any errors been detected ??
  no
    Set error report to no errors
    Return error report
  yes
    ?? Is error class an interface failure or all nodes failed ??
      yes
        Set error report to indicate root link failure and current channel
        Return error report
      no
        ?? Is error class a babbler ??
          yes
            Set error report to indicate a babbler
            Return error report
          no
            Initialize error analysis variables:
            Set failed node count to zero
            Set failed node set to empty
            Set nodes visited to empty
            Verifiy set is empty
            Clear failed node list
            for << each transaction in status collection chain >>
              ?? Is the transaction selected ??
                no
                yes
                  ?? Did the transaction have an error ??
                    no
                    yes
                      Increment failed node count
                      Add node to failed node set
                      Put node on failed node list
            ?? Does failed node count equal 1 ??
              yes
                Error report indicates single node failure and identifies failed node
                Return error report
              no
                Find root of failed tree
                ?? Root found ??
                  no
                    Error report indicates analysis unsucessful
                  yes
                    Add Root to verify set
                    Zero recursion count
                    ?? Is there a failed tree??
                      no
                      yes
                        ?? Does verify set equal failed node set ??
                          no
                          yes
                            Error Report shows link failure and associated information
                Return error report
```

Software Specification Reference Number: 3.1.2.2

Subprogram: DATA ANALYSIS
Inputs: I/O network identifier
current channel
node configuration
network topology
Outputs: error report record

```
Clear bad port flag
Error report is set to no errors
For << each node response >>
    ?? Any transmission errors detected on this response ??
    yes
    no
        ?? Did node status indicate a valid frame received
           on an idle port adjacent to a node ??
        no
        yes
            ?? Is this the only node found in this condition so far ??
            no
                Raise undiagnosable error
            yes
                Set bad port flag
                Assign fields of fault analysis
?? Is the bad port flag set ??
no
yes
    Assign fields of error report
Return error report
```

Subprogram: TRANSIENT ANALYSIS
Inputs: I/O network identifier
first report
node configuration
current channel
Outputs: error report
transient flag

```
Collect node status
?? Do results of second status collection agree with results of first ??
yes
    Errors are not transient
no
    ?? Were any errors detected in second status collection ??
    no
        Errors are transient
    yes
        Collect node status again
        ?? Third status collection results agree with second ??
        yes
            Errors are not transient
        no
            Errors are transient
```

Software Specification Reference Number: 3.1.2.2

Subprogram: FAILED TREE
Inputs: root
Outputs: boolean flag

```
Increment recursion count
?? Has this node been visited or the maximum
   recursion count been exceeded ??
yes
    Raise
    undiagnosable
    error
no
    Add node to nodes visited set
    For << each port of this node >>
        ?? Is this port configured inboard or idle ??
        yes
        no
            ?? Is the element adjacent to this port a node ??
            no
            yes
                Add this node to verify set
                ?? Is this adjacent node in the failed node set ??
                yes
                    ?? Failed tree of this
                       adjacent node ??
                    yes
                    no
                        Return false
                no
                    Return false
    Return true
```

Subprogram: ROOT OF FAILED TREE
Inputs: none
Outputs: node number type

```
Set found to FALSE
for << each failed node in failed node list >>
    Identify inboard port of this failed node
    Find type of element adj to inboard port of failed node
    ?? Element adjacent to this port a node ??
    no                               | yes
    ?? Element adj to this           | ?? Response from this adj node had error ??
       port a GPC ??                 |
    no    | yes                      | no                              | yes
          | ?? Only root found so far ??                               |
          | yes                          | no                          |
          | Set root to current          | Raise                       |
          |     failed node              | undiagnosable               |
          | Set found to TRUE            | error                       |
          |                              | exit                        |
?? Found a failed root ??
yes
    Return value of root
no
    Raise undiagnosable error
```

Software Specification Reference Number: 3.1.2.2

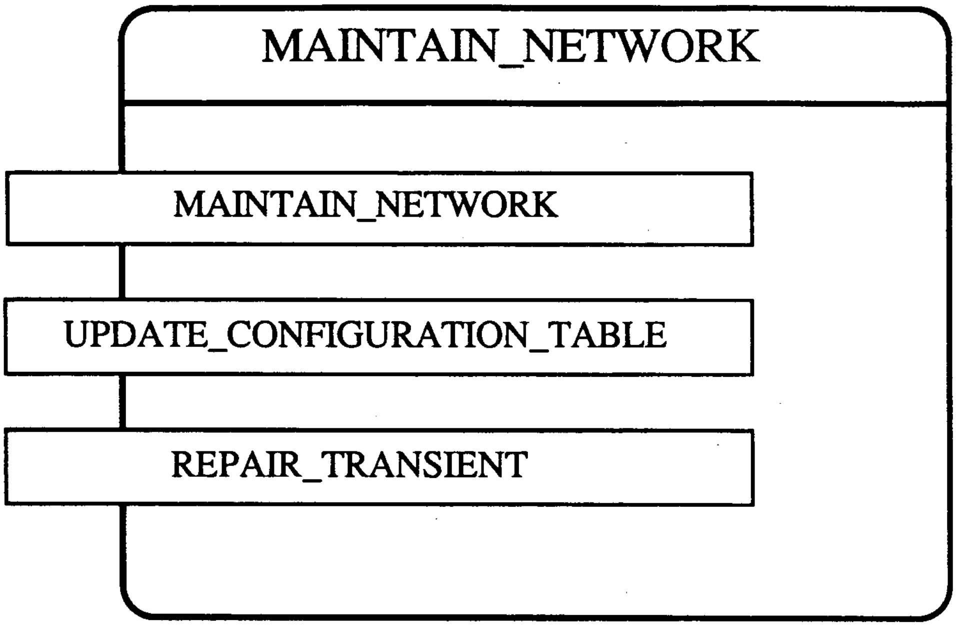
Subprogram: INBRD_PORT
Inputs: node_number
Outputs: port_number

```
for << each port >>
    ?? Is the node configuration for this port an inboard port ??
    no:
    yes:
        Set found to TRUE
        Return the port
?? Node has no inboard port ??
no:
yes:
    Log the error
```

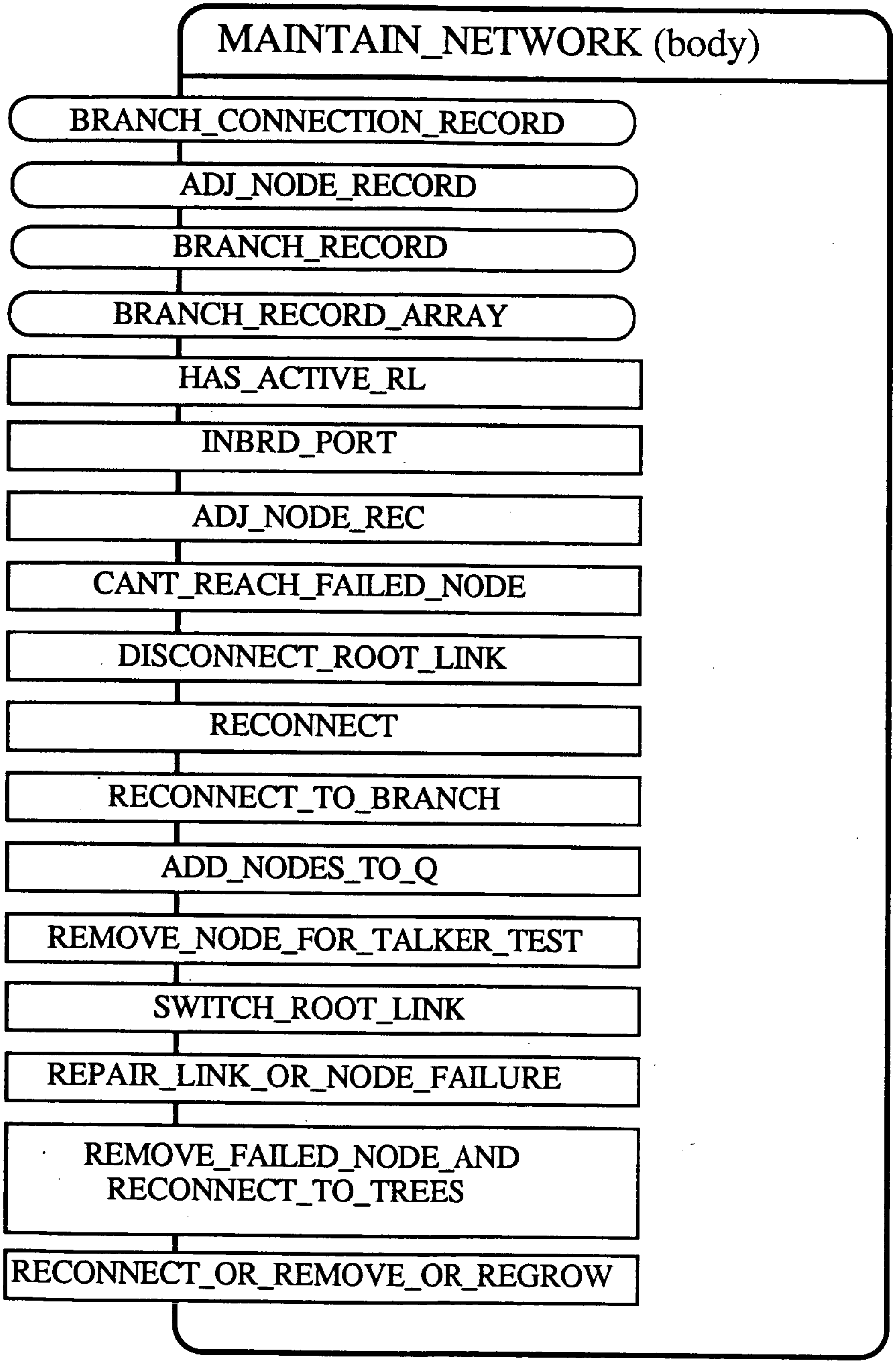
Subprogram: EQUAL_NET_STATUS
Inputs: first report
second report
Outputs: boolean flag

```
Initialize equal to FALSE
?? Do the first and second reports both show an interface failure
   and then have their attributed_to field set to the same value ??
yes:
    Set equal to TRUE
no:
    ?? Do the first and second reports both not report an interface
       failure and both report a babbler dectected and then both
       have their detected field set to the same value??
    yes:
        Set equal to TRUE
    no:
        ?? Do the first and second records both
           not report a babbler dectected ??
        no:
        yes:
            for << each node >>
                ?? Is the node for the first record selected and then
                   the had_error field of the first and second
                   record are not set to the same value??
                no:
                yes:
                    Set equal to FALSE
                    Exit the loop
Return equal
```

Software Specification Reference Number: 3.1.2.2



Software Specification Reference Number: 3.1.2.3



Software Specification Reference Number: 3.1.2.3

Subprogram: MAINTAIN_NETWORK

Inputs:	network topology
	i/o network identifier
	current channel
	error report
	root links
	root link history
	network status
	node configuration
Outputs:	node configuration
	network status
	root link history
	current channel

```
Initialize reconfig_successful to FALSE
Initialize repair_attempts to zero
while << reconfigure is unsucesful and repair_attempts
         is less than max_repair_attempts >>
    Increment repair_attempts
    ?? Error analysis successful ??
    no
        Regrow network
        Set reconfigure success to TRUE
    yes
        ?? Fault type is babbler ??
        yes
            Select fast grow option
            Regrow network
            Set reconfigure success to TRUE
        no
            ?? Fault type is no fault ??
            yes
                Set reconfigure success to TRUE
            no
                ?? Fault type is root link failure ??
                yes
                    Call Switch root links
                no
                    ?? Fault type is link or node failure ??
                    yes
                        Call Repair link or node failure
                    no
                        ?? Fault type is talks out disabled port
                           or bad address or configure data ??
                        yes
                            Call Remove node and reconnect to trees
                        no
                            ?? Fault type is single node failure ??
                            no
                            yes
                                Call Reconnect or remove or regrow
?? Unexpected event during reconfiguration or network
   has no active root link and regrowth not tried ??
no
yes
    Select fast grow option
    Regrow network
```

Software Specification Reference Number: 3.1.2.3

Subprogram: REPAIR_TRANSIENT
Inputs: changes
Outputs: changes

```
No transient analysis logic in place yet hence no changes to net will occur
Set changes to FALSE
```

Subprogram: UPDATE CONFIGURATION TABLE
Inputs: network topology
first node
new inboard port
node configuration
Outputs: node configuration

```
Set done updating to FALSE
Set current node to first node
while << not done updating >>
    Set found inboard port to FALSE
    for << each port in current node >>
        ?? Is this port the inboard port of current node ??
        no
        yes
            Set found inboard port to TRUE
            Set value of old inboard port to inboard port
            exit
    Set the configuration of the new inboard port of current node to inboard
    ?? Inboard port found in current node??
    no
        Set done updating to TRUE
    yes
        ?? Adjacent element is GPC ??
        yes
            Set config of old inboard port of current node to idle port
            Set done updating to TRUE
        no
            Set config of old inboard port to outboard
            Using Network topology assign values to current node and its new inboard port
```

Software Specification Reference Number: 3.1.2.3

Subprogram: HAS_ACTIVE_RL
Inputs: interface status
Outputs: boolean flag

```
for << each channel >>
    ?? Is the interface status of this channel set to active ??
    yes
        Return TRUE
    no
Return FALSE
```

Subprogram: INBRD_PORT
Inputs: node number type
Outputs: port number type

```
for << each port in port number type >>
    ?? Is the configuration of this port inboard ??
    no
    yes
        Set found to TRUE
        Return the port
?? No inboard port was found ??
no
yes
    Log the error
    Raise a constraint error
```

Subprogram: ADJ_NODE_REC
Inputs: target node record
Outputs: adjacent node record

```
Look up the target node in the network topology
Return the node number and port number of the
    node adjacent to this port of the target node
```

Subprogram: CANT_REACH_FAILED_NODE
Inputs: failed node
Outputs: boolean flag

```
for << each port >>
    ?? Is the element adjacent to this port a node
       and the status of the port active ??
    no
        Return TRUE
    yes
Return FALSE
```

Subprogram:DISCONNECT_ROOT_LINK
Inputs: root link to disconnect
Outputs: none

```
Set up command to root node to disable port facing IOS
Call configure_nodes with the no log option
Set the status of that root link to failed ios
```

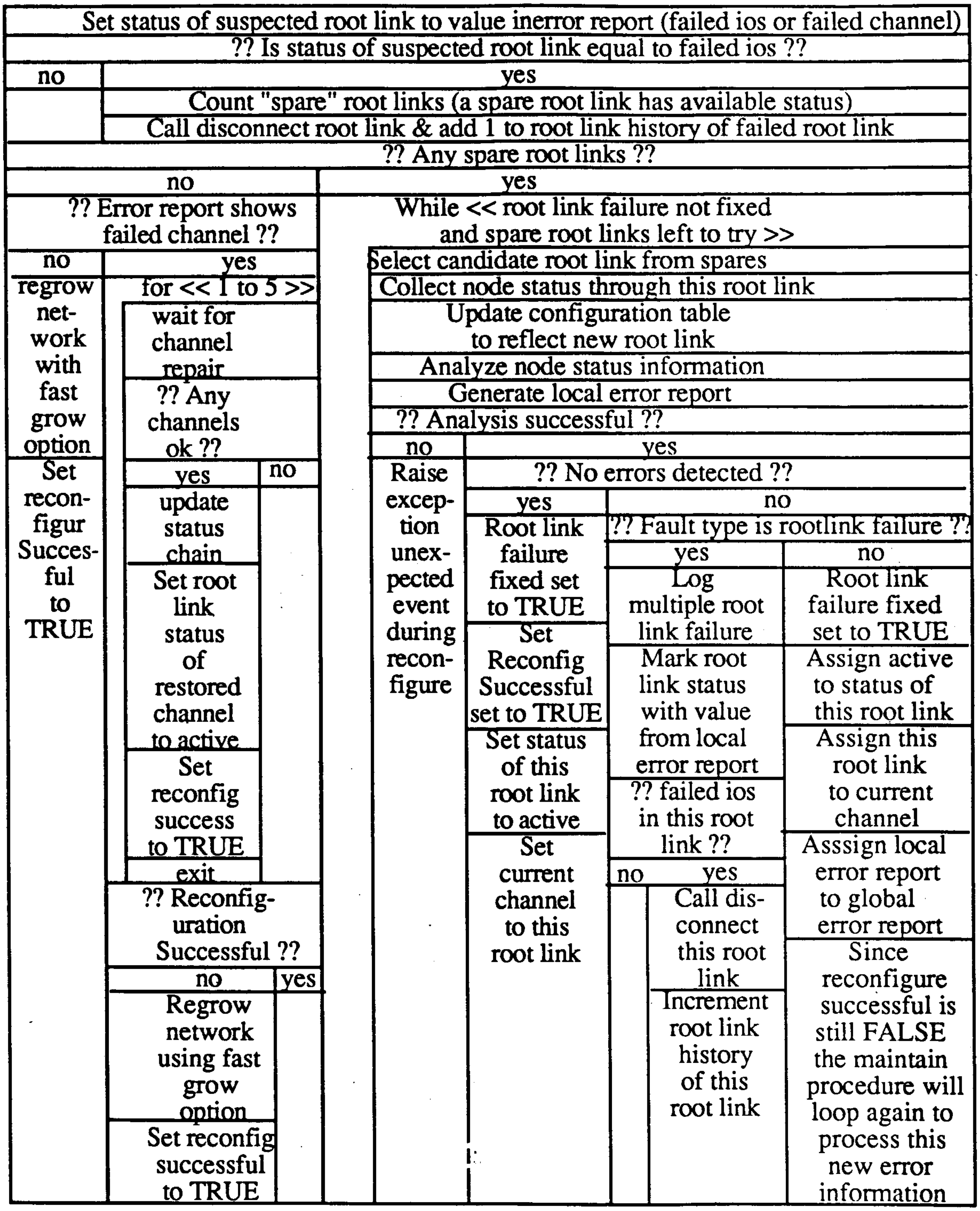
Software Specification Reference Number: 3.1.2.3

Subprogram: RECONNECT TO BRANCH
Inputs: my branch
failed node set
root of failed tree
port facing this branch on root of failed tree
Outputs: result of reconnection

```
Set repair complete to false
Set branch reconnected to false
while << branch is not reconnected >>
    ?? Any nodes left to try ??
    no:
    yes:
        Select target node from branch
        for << each port of target node >>
            ?? Is this port an idle port adjacent to an acitve node which
               is not in the failed node set ??
            no:
            yes:
                Set spawning node to adjacent node
                Set up command to spawning node to enable
                    port adjacent to target node
                Set up command to target node to enable
                    port adjacent to spawning node
                Set up command to node at root of failed tree
                    to disable port adjacent to failed link
                Select contention option for chain execution
                Call configure nodes
                ?? Summarized errors in error report shows ?
                Babbler Detected / Interface failure / All failed:
                    Deselect contention option and retry once
                    Summarize errors in second try
                Any errors:
                    retry once
                    Summarize errors in second try
                No errors:
                ?? Error summary shows ??
                No errors:
                    Mark status of ports active
                    Set config of target port to inboard
                    Set config of spawning port to outboard
                    Call update config table
                    Collect node status
                    ?? Any errors ??
                    yes:
                    no:
                        Set repair complete to TRUE
                    Set connection to branch to TRUE
                Babbler or error only on root of failed tree:
                    Disconnect root of failed tree from this branch
                    Set status of ports to failed
                    Set configuration of ports to idle port
                    Reenable link between spawning and target nodes
                    ?? Link enabled ??
                    yes:
                        Set status of ports to active
                        Set config of target port to inboard
                        Set config of spawning port to outboard
                        Set connection to branch to TRUE
                    no:
                Any other error:
                    Disconnect link between target and spawning node
                    Set status of ports to failed
                    Set configuration of ports to idleport
                    Log event
                I/F Error:
                    Log event
                    Raise unexpected event during reconfiguration attempt
```

Software Specification Reference Number: 3.1.2.3

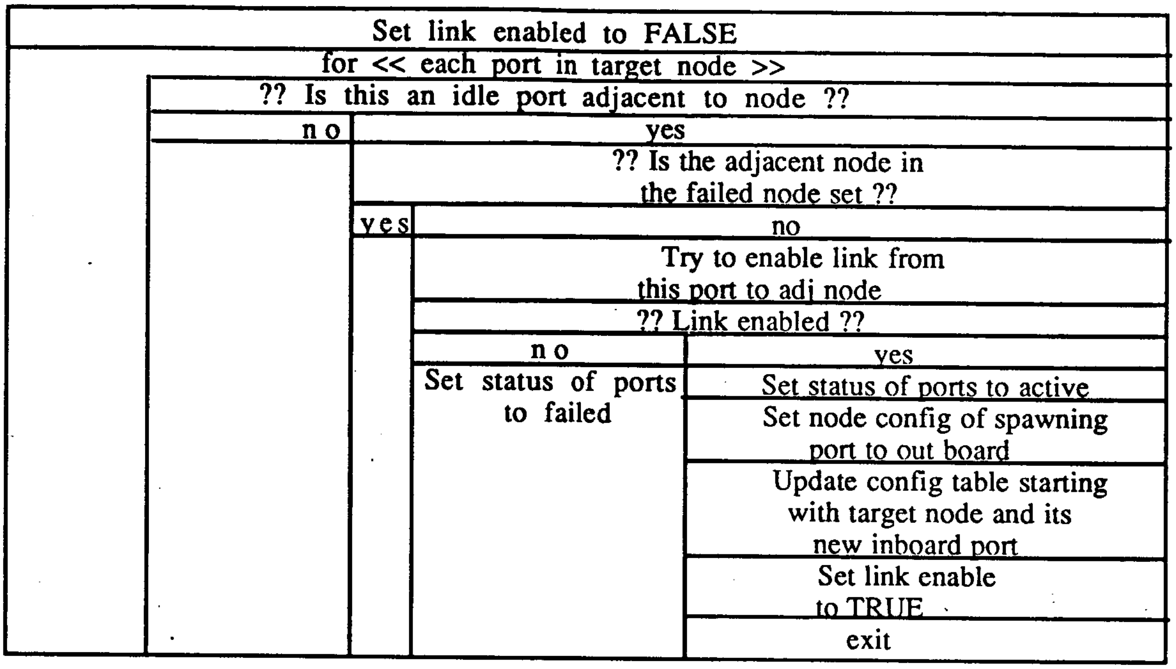
Subprogram: SWITCH_ROOT_LINK
Inputs: error reports
Outputs: none



```
Set status of suspected root link to value inerror report (failed ios or failed channel)
?? Is status of suspected root link equal to failed ios ??
  no
  yes
    Count "spare" root links (a spare root link has available status)
    Call disconnect root link & add 1 to root link history of failed root link
?? Any spare root links ??
  no
    ?? Error report shows failed channel ??
      no
        regrow network with fast grow option
        Set reconfigur Succesful to TRUE
      yes
        for << 1 to 5 >>
          wait for channel repair
          ?? Any channels ok ??
            yes
              update status chain
              Set root link status of restored channel to active
              Set reconfig success to TRUE
              exit
            no
        ?? Reconfiguration Successful ??
          no
            Regrow network using fast grow option
            Set reconfig successful to TRUE
          yes
  yes
    While << root link failure not fixed and spare root links left to try >>
      Select candidate root link from spares
      Collect node status through this root link
      Update configuration table to reflect new root link
      Analyze node status information
      Generate local error report
      ?? Analysis successful ??
        no
          Raise exception unexpected event during reconfigure
        yes
          ?? No errors detected ??
            yes
              Root link failure fixed set to TRUE
              Set Reconfig Successful set to TRUE
              Set status of this root link to active
              Set current channel to this root link
            no
              ?? Fault type is rootlink failure ??
                yes
                  Log multiple root link failure
                  Mark root link status with value from local error report
                  ?? failed ios in this root link ??
                    no
                    yes
                      Call disconnect this root link
                      Increment root link history of this root link
                no
                  Root link failure fixed set to TRUE
                  Assign active to status of this root link
                  Assign this root link to current channel
                  Asssign local error report to global error report
                  Since reconfigure successful is still FALSE the maintain procedure will loop again to process this new error information
```

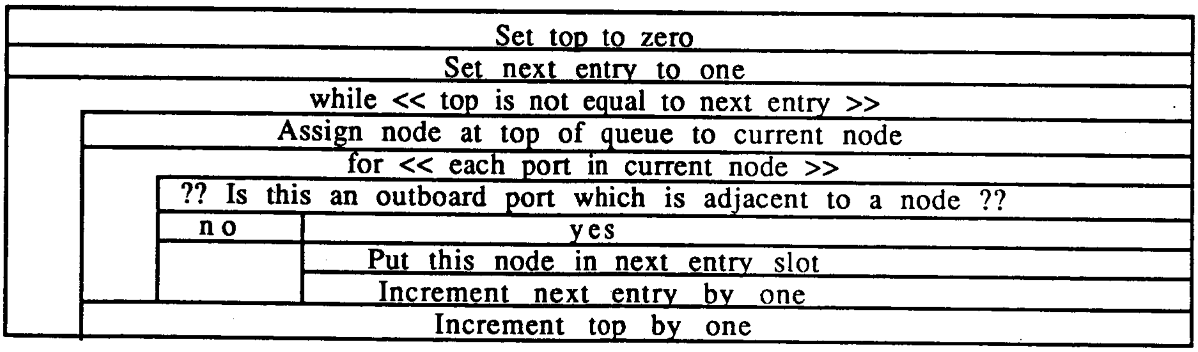
Software Specification Reference Number: 3.1.2.3

Subprogram: RECONNECT (to target node)
Inputs: target
failed node set
Outputs: link enabled



```
Set link enabled to FALSE
for << each port in target node >>
    ?? Is this an idle port adjacent to node ??
    no
    yes
        ?? Is the adjacent node in the failed node set ??
        yes
        no
            Try to enable link from this port to adj node
            ?? Link enabled ??
            no
                Set status of ports to failed
            yes
                Set status of ports to active
                Set node config of spawning port to out board
                Update config table starting with target node and its new inboard port
                Set link enable to TRUE
                exit
```

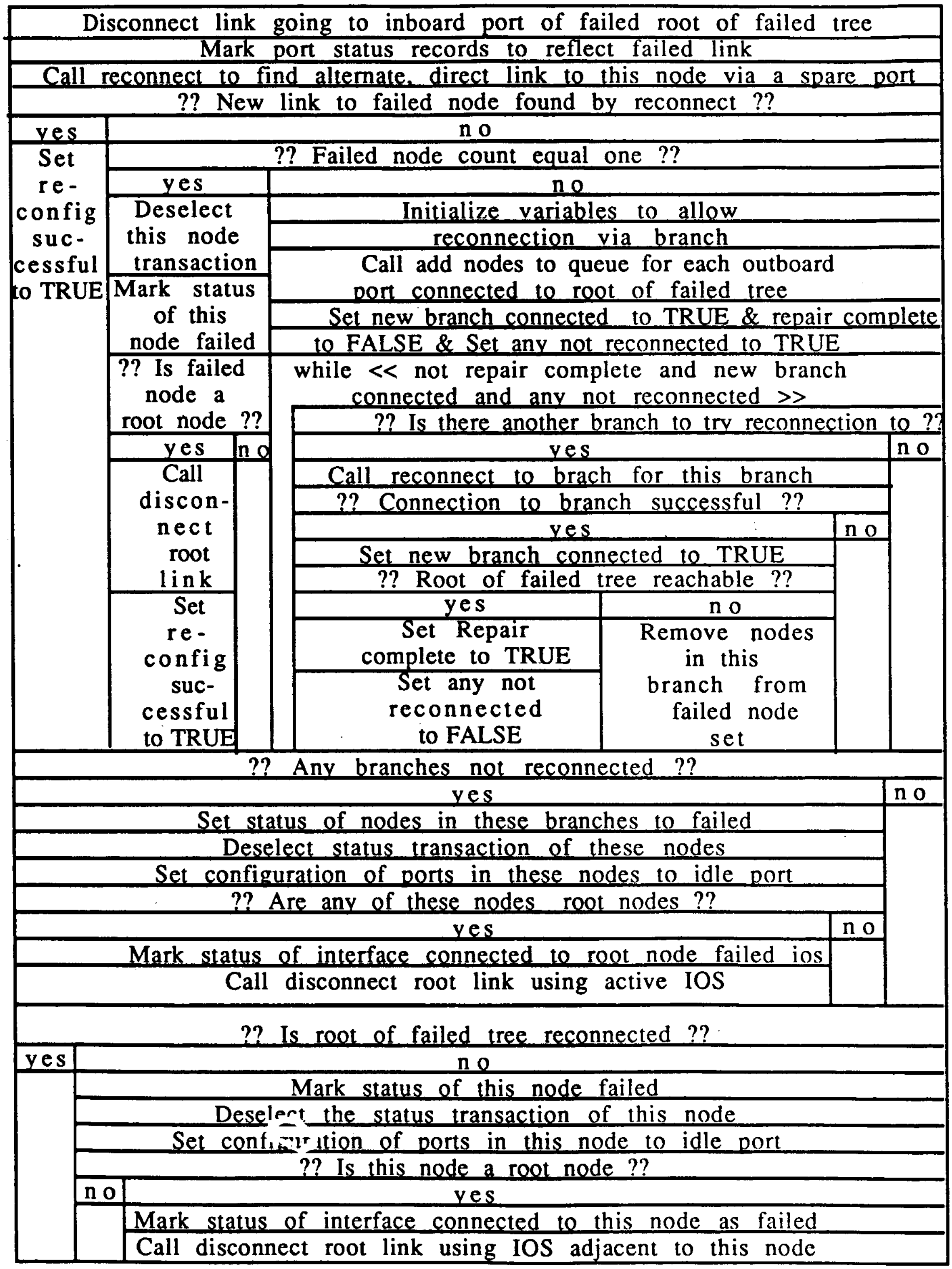
Subprogram: ADD NODES TO QUEUE
Inputs: next entry
Outputs: target queue



```
Set top to zero
Set next entry to one
while << top is not equal to next entry >>
    Assign node at top of queue to current node
    for << each port in current node >>
        ?? Is this an outboard port which is adjacent to a node ??
        no
        yes
            Put this node in next entry slot
            Increment next entry by one
    Increment top by one
```

Software Specification Reference Number: 3.1.2.3

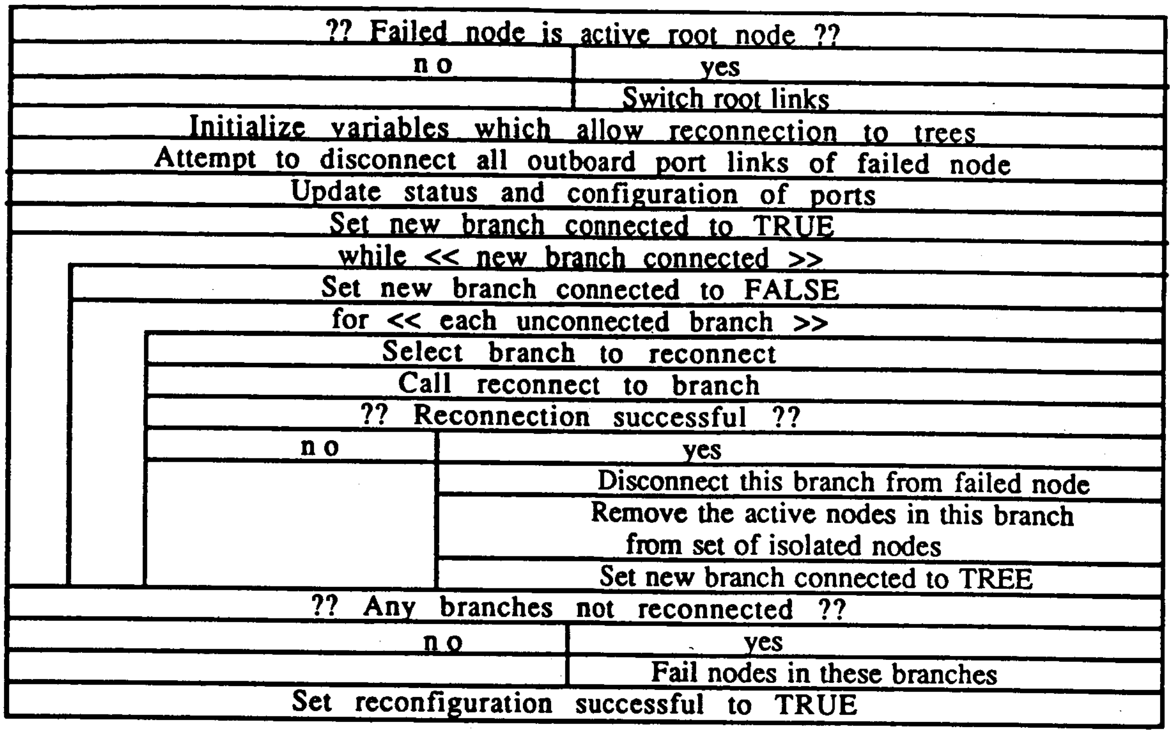
Subprogram: REPAIR_LINK_OR_NODE_FAILURE
Inputs: none
Outputs: none



```
Disconnect link going to inboard port of failed root of failed tree
Mark port status records to reflect failed link
Call reconnect to find alternate, direct link to this node via a spare port
?? New link to failed node found by reconnect ??
yes
    Set re-config successful to TRUE
no
    ?? Failed node count equal one ??
    yes
        Deselect this node transaction
        Mark status of this node failed
        ?? Is failed node a root node ??
        yes
            Call disconnect root link
            Set re-config successful to TRUE
        no
    no
        Initialize variables to allow reconnection via branch
        Call add nodes to queue for each outboard port connected to root of failed tree
        Set new branch connected to TRUE & repair complete to FALSE & Set any not reconnected to TRUE
        while << not repair complete and new branch connected and any not reconnected >>
            ?? Is there another branch to try reconnection to ??
            yes
                Call reconnect to brach for this branch
                ?? Connection to branch successful ??
                yes
                    Set new branch connected to TRUE
                    ?? Root of failed tree reachable ??
                    yes
                        Set Repair complete to TRUE
                        Set any not reconnected to FALSE
                    no
                        Remove nodes in this branch from failed node set
                no
            no
?? Any branches not reconnected ??
yes
    Set status of nodes in these branches to failed
    Deselect status transaction of these nodes
    Set configuration of ports in these nodes to idle port
    ?? Are any of these nodes root nodes ??
    yes
        Mark status of interface connected to root node failed ios
        Call disconnect root link using active IOS
    no
no
?? Is root of failed tree reconnected ??
yes
no
    Mark status of this node failed
    Deselect the status transaction of this node
    Set configuration of ports in this node to idle port
    ?? Is this node a root node ??
    no
    yes
        Mark status of interface connected to this node as failed
        Call disconnect root link using IOS adjacent to this node
```

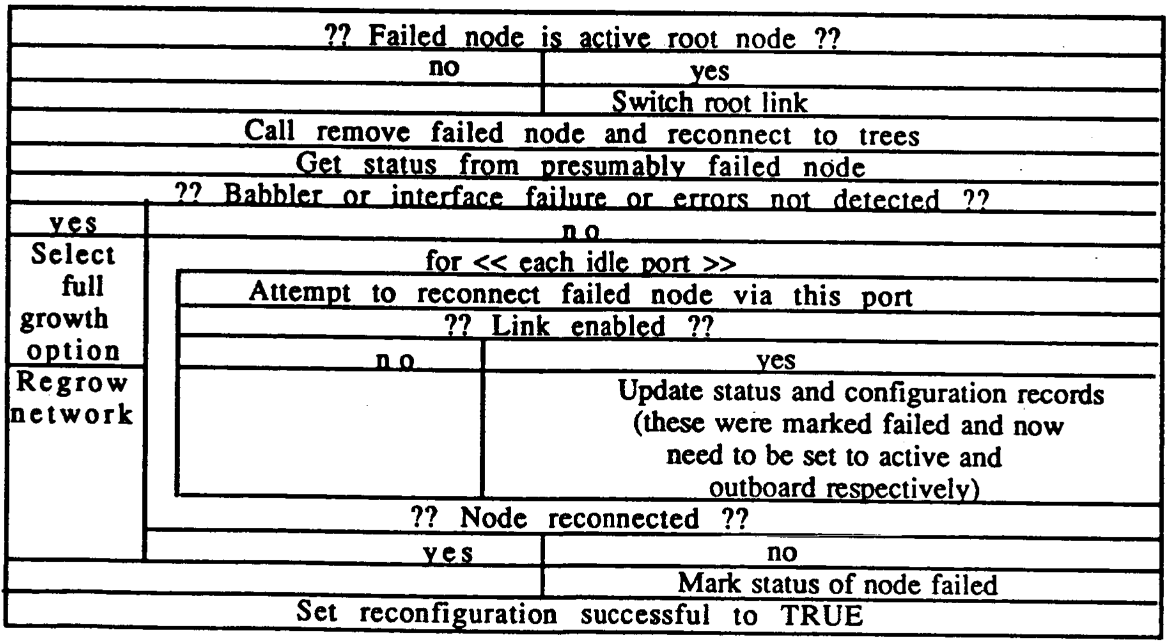
Software Specification Reference Number: 3.1.2.3

Subprogram: REMOVE FAILED NODE AND RECONNECT TO TREES
Inputs: none
Outputs: none



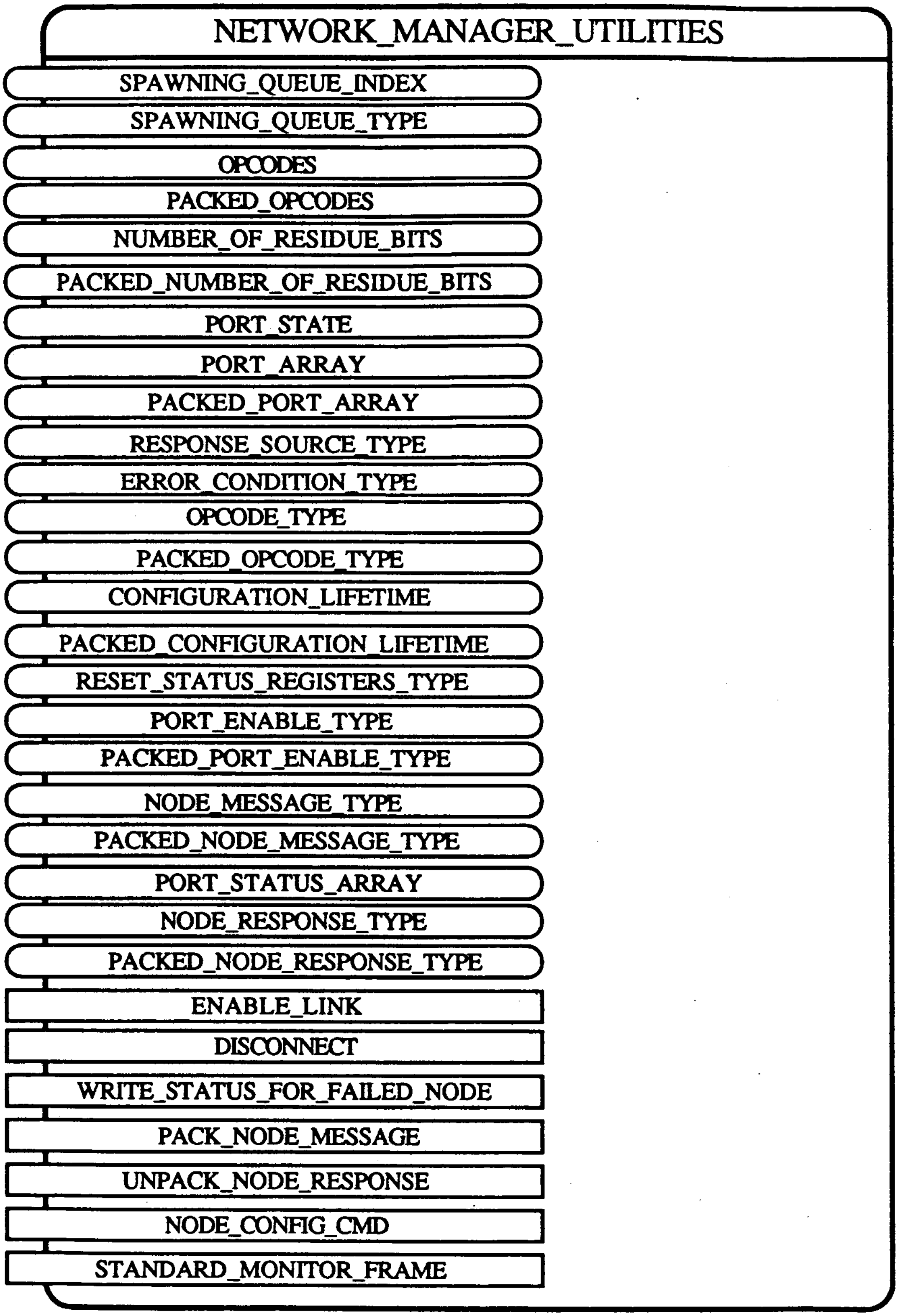
```
?? Failed node is active root node ??
    no
    yes
        Switch root links
Initialize variables which allow reconnection to trees
Attempt to disconnect all outboard port links of failed node
Update status and configuration of ports
Set new branch connected to TRUE
while << new branch connected >>
    Set new branch connected to FALSE
    for << each unconnected branch >>
        Select branch to reconnect
        Call reconnect to branch
        ?? Reconnection successful ??
            no
            yes
                Disconnect this branch from failed node
                Remove the active nodes in this branch
                  from set of isolated nodes
                Set new branch connected to TREE
?? Any branches not reconnected ??
    no
    yes
        Fail nodes in these branches
Set reconfiguration successful to TRUE
```

Subprogram: RECONNECT OR REMOVE OR REGROW
Inputs: none
Outputs: none



```
?? Failed node is active root node ??
    no
    yes
        Switch root link
Call remove failed node and reconnect to trees
Get status from presumably failed node
?? Babbler or interface failure or errors not detected ??
    yes
        Select full growth option
        Regrow network
    no
        for << each idle port >>
            Attempt to reconnect failed node via this port
            ?? Link enabled ??
                no
                yes
                    Update status and configuration records
                    (these were marked failed and now
                    need to be set to active and
                    outboard respectively)
        ?? Node reconnected ??
            yes
            no
                Mark status of node failed
Set reconfiguration successful to TRUE
```

Software Specification Reference Number: 3.1.2.3



Software Specification Reference Number: 3.1.2.3

Subprogram: ENABLE_LINK
Inputs: i/o network identifier
network topology
current channel
contention option
from_node
outboart port
to_node
inboard port
max tries
node configuration
link_enabled
chain report
Outputs: link_enabled
chain report

```
Set up commands for outboard and inboard ports
Repeat until link enabled equals TRUE or tries exceeds max tries
    Link enabled := FALSE and configure nodes to enable link
    ?? babbler detected or transmission errors ??
    yes
        Disconnect to node and from node
    no
        Test for babbler
        ?? Was the babbler detected ??
        no
            Set link enabled
        yes
            Disconnect to_node & from_node
```

Software Specification Reference Number: 3.1.3.1,3.1.3.3

Subprogram: DISCONNECT
Inputs: i/o network identifier
network topology
node configuration
from_node
outboard port
to_node
inboard port
current channel
contention option
Outputs: none

```
Set up a command to to_node to disable inboard port
Set up a command to from_node to disable outboard port
Call configure nodes to send the commands
```

Subprogram: WRITE_STATUS_FOR_FAILED_NODE
Inputs: failed node
network topology
network status
Outputs: network status

```
Set the node status to failed
for < each port >
    Set the port status to failed
    ?? case adjacent element ??
        my gpc channel:
            Set interface status of adj. IOS to failed IOS
        node:
            Set the adj. port status to failed
        others:
            null
```

Software Specification Reference Number: 3.1.3.1 3.1.3.3

Subprogram: NODE_CONFIGURATION_COMMAND
Inputs: configuration lifetime
node address
desired port configuration
Outputs: node message

```
Assign node address field of node message the value of address
Assign node field of opcode field of node message a value of reconfigure ports
Assign status message field of opcode field of node message a value of status register
Assign error field of opcode field of node message a value of valid
Assign residue bytes field of opcode field of node message a value of three
Assign lifetime field of port enable field of node message a value of configuration lifetime
Assign clear status field of port enable field of node message a value of clear
For << each port >>
    ?? Is the desired port configuration of this port idle port ??
        yes
            Turn corresponding port enable off
        no
            Turn corresponding port enable on
Return node message
```

Subprogram: STANDARD_MONITOR_FRAME
Inputs: node address
Outputs: node message

```
Assign node address field of node message the value of address
Assign mode field of opcode field of node message the value of report status
Assign the status message field of opcode field of node message the value of status register
Assign the residue bit field of opcode field of node message the value of three
Assign the lifetime field of port enable field of node message the value of once only
Assign the clear status field of port enable field of node message the value of clear
Assign each port field of port enable field of node message the value of off
Return node message
```

Software Specification Reference Number: 3.1.3.2

Subprogram: CODED_NODE_ADDRESS
Inputs: node address
Outputs: encoded node address

Initialize max value to 255
Return (max value minus node addresss)

Subprogram: MESSAGE_SUM_CHECK
Inputs: node message
Outputs: node check sum

Initialize temp sum to node address from node message	
Add encoded address to temp sum	
?? Is temp sum greater than or equal to modulus ??	
no	yes
	Subtract modulus from temp sum
Add opcode to temp sum	
?? Is temp sum greater than or equal to modulus ??	
no	yes
	Subtract modulus from temp sum
Add port enable to temp sum	
?? Is temp sum greater than modulus ??	
no	yes
	Subtract modulus from temp sum
Return modulus minus temp sum	

Subprogram: PACK_NODE_MESSAGE
Inputs: node message
Outputs: packed node message

Convert internal representation of fields of node message to bit mapped representation required by node
Call message sum check to append check sum
Return packed representation of node message

Subprogram: UNPACK_NODE_RESPONSE
Inputs: packed node response
Outputs: node response

Convert bit mapped representation of fields of packed node response to internal representation
Return internal representation of node response

Software Specification Reference Number: 3.1.3.2

5.0 CONCLUSIONS AND RECOMMENDATIONS

The Advanced Information Processing System (AIPS) Input/Output Network Software has been designed, implemented and tested on the centralized configuration of the AIPS engineering model. This network management software manipulates the large number of possible interconnections between the circuit switched nodes to maximize the system's overall reliability and survivability. The responsibilities of this software include the following: initial growth of a network, establishing active paths to each functional node; periodic testing of each node in order to determine if the node is accessible; detecting faults in the network; periodic cycling of spare links in the network to ensure that they are fault free; reconfiguring the network to isolate faulty components; and re-establishing connections to nodes which have been repaired. It is composed of 19,994 lines of Ada source code. The source code includes 8,604 Ada statements (which may take more than one line) and comment statements. The source code and global variables use 370,748 bytes of memory.

5.1 Testing Of Network Manager Software

Initial testing of the network management software was done by randomly injecting faults into the links, nodes, root nodes and IOSs. The network status display and error logs were used to monitor these tests. The software correctly identified and reconfigured the network in all tests. Performance and reliability metrics for the centralized AIPS system have not been measured as yet. They need to be gathered and evaluated under fault free and degraded conditions of the network.

5.2 Future Work

After the performance metrics are gathered and evaluated for the network hardware and software, the timing or bandwidth bottlenecks will be uncovered. Additionally, there are known areas where software performance can be improved. For example, the most time consuming identification procedures are those that require a regrow of the network, such as the identification of the faulty node when a node fails active. The babbler and the node which responds out of turn are two examples of active node failures. The latter example, a node which answers to an address other than its own, requires network regrowth with full diagnostics. Other failures that use the regrowth with diagnostic testing algorithm for identification and reconfiguration are those faults whose source is not identified during periodic status collection.

If the fault is a babbler, the network is regrown without diagnostic testing since the detection and isolation of a babbler does not require these tests. A babbler is an active fault and includes a stuck on high condition detected by the IOS at its receiving interface to the network. In the present algorithm, the cost of regrowing a network of N nodes in the presence of a babbler is N + P chains, where P is the number of spare ports on the faulty node that must be tried until a non-faulty one is found. A faster algorithm could be designed which would reduce this cost in the case of networks that are either maximally branching or fully linear. In the latter case a binary search could be used. The node in the middle of the bus would have its outboard port disabled and the location of the babbler would be deduced from the continued presence or absence of babbler symptoms after this reconfiguration. For the maximally branching network, a similar search could be conducted on each outboard branch of a node. If disabling the port which leads to a branch eliminates the babbler's symptoms, then that port is the gateway to the branch of the network containing the babbler. However, the network configuration may be a mix of these two basic patterns. The cost of finding the babbler is then not only a function of the number of chains necessary to identify and isolate the babbler, but also the cost of deciding on which type of search to employ. The decision as to which algorithm is least expensive depends on the number of nodes in the network. More analysis of the problem is needed to make an informed choice and develop an algorithm that meets the performance and reliability requirements of the system.

Regrowth with testing is performed in the cases where the network manager software is unable to identify the type of fault or the faulty component. Examples of these are faults in the switching logic which are configuration dependent and some types of the talk out of turn faults. (In some cases, the talk out of turn node is identified by the manager and it is simply removed from the network.) In the case where a talks out of turn node is not identified by the manager, the time consuming regrowth with testing algorithm is used for fault identification. If an identification algorithm were developed that could always determine when a fault is due to a node responding out of turn, then the regrow with testing algorithm would not be necessary for this type of failure. Instead, the manager could respond with a simpler reconfiguration strategy.

The use of the regrowth and other time consuming algorithms for fault diagnosis and repair can require inordinate computational intensity which may take the network off-line for more than one I/O cycle. This potential lack of availability might require a parallel network to meet the performance and reliability requirements of the application. These adversities arise primarily because the sender of a transmission cannot be identified with certainty, a relayer of a transmission can mutate a message without certainty of detection, and a noisy node or babbler cannot be identified without lengthy diagnosis and reconfiguration procedures. In addition, unlike other CSDL-designed AIPS building blocks, the networks are not Byzantine Resilient, nor can they be made so without massive network replication. Therefore, they are not demonstrably resilient to malicious faults, so the network validation process does not benefit from the theoretical rigor of the Byzantine Resilience approach to fault tolerance. An appropriate communication protocol might assist in reducing the computational and communication overhead involved in diagnosing and repairing network faults, and might also allow the construction of Byzantine Resilient networks. To determine the feasibility of such an approach, a study is needed to examine several potentially applicable authentication schemes and ascertain their computational and communication overhead, probability of failure, and other performance measures.

Transient fault analysis is another area requiring further research and development. In particular, it is difficult to distinguish between an intermittent failure and a transient. At present, automatic retries of chains are used to identify transient faults, but nothing is done for the intermittent fault, which will appear to the manager as a transient. To deal with intermittent faults a system of demerits could be employed. When a fault is detected, but does not reappear in the retry, a demerit could be charged to the hardware causing the error, if that can be determined, or to the entire network if a more specific cause cannot be found. If the demerit is assigned to the entire network and the network eventually falls below a certain threshold, sections of the network could be taken off line, one at a time, in order to isolate the fault. Another area requiring further research is determining the amount of time to wait between retries. If the chain is retried immediately as in the present algorithm, a transient of a long duration is declared a permanent fault. Since no recovery is attempted until the faulty component is repaired, that resource is lost. Since 50 to 80 per cent of faults are typically transients, the network fault detection algorithms need to be expanded to include transient/intermittent fault analysis and tested with the actual hardware by simulating intermittent and transient faults.

Networks are grown at initialization time from the network database. At present that database is static. An algorithm should be developed to support a dynamic database. In such a case the software could determine the actual physical network topology dynamically. If a new network is connected after system startup, it then would be possible to add it to the system with an operator command. A dynamic database would also allow a graceful addition of new nodes and links to an existing network.

Finally, algorithms need to be developed for handling errors which are observed by one site of a regional network, but not at the site hosting the Network Manager. This is especially important for the Inter-Computer Network Manager.

6.0 REFERENCES

1. J.H. Lala, A. Ray, R. Harper, "A Fault and Damage Tolerant Network for an Advanced Transport Aircraft," American Automatic Control Conference, San Diego, CA, June, 1984.
2. G. Nagle, "An Ada Implementation of the Network Manager for the Advanced Information Processing System," The First International Conference on Ada Programming Language Applications for the NASA Space Station, Houston, TX, June, 1986.
3. G. Booch, Software Engineering with Ada®, the Benjamin/Cummings Publishing Co., Inc., Menlo Park, CA, 1983.
4. I. Nassi and B. Schneiderman, "Flowchart Techniques for Structured Programming", Department of Computer Science, State University of New York at Stony Brook, New York, August, 1973.

APPENDIX A: GLOSSARY OF I/O NETWORK TERMS

Network Hardware

Node: a circuit switching device with 5 ports which can each be independently enabled or disabled. An enabled port retransmits the logical OR of all data which has been received by any other enabled port. The retransmission is carried out with minimal delay, nominally one half the period of the transmission clock.

Device Interface Unit. (DIU): The smallest unit addressable by an application on an I/O network. DIUs may be single devices (such as sensors or actuators) or collections of such devices.

IOP: I/O Processor.

CP: Computational Processor.

GPC: General Purpose Computer consisting of an IOP, a CP, and their interfaces to I/O and IC networks.

I/O Sequencer (IOS): A state machine whose function is to conduct the physical aspects of communication on an I/O network for a GPC. The IOS communicates with one channel of. a GPC by means of a Dual Ported Memory. The IOS executes a program which has been stored in (DPM) by the IOP. Part of the DPM behaves as a set of control and status registers for the IOS. Once a program has been stored in the DPM, communication between the IOS and GPC can be conducted by means of the control and status registers. It is possible to store programs in the DPM for future execution and then in real time it is only necessary to update the data required by these programs. Input data must be exchanged across redundant channels for source congruency and output data must be voted to provide fault masking. Each IOS is a simplex device which performs its function asynchronously from other IOSs and from the GPC to which it is connected.

Network Interface: the hardware involved in the connection between a GPC and a network. It consists of an IOS, 8 K bytes of dual ported memory, and a link (called the root link) connecting the IOS to a network node (called the root node).

I/O Network: a set of nodes and DIUs which are physically interconnected.

I/O Network Topology: The specific interconnections among the nodes, GPCs and DIUs in an I/O Network.

Virtual Bus: A network whose nodes have been configured to allow communication between a GPC and DIUs or nodes on that network to emulate communication on a serial bus.

Network Classification

I/O Service: A logical organization imposed on I/O network use. A service may be designated as Regional or Local.

Regional I/O Service: I/O activity conducted on a single I/O Network which is shared among several GPCs. Since only one GPC may use the network at any given time, GPCs must contend for use of the network.

Local I/O Service: I/O activity conducted on an I/O network which is used exclusively by one GPC. If an I/O network which is part of a Local I/O Service is physically connected to more than one GPC, exactly one of those GPCs will be included in the service at any given time. A change in the GPC included in the service constitutes a function migration.

Redundant I/O Network: a set of I/O networks connected to the same GPC. Each network in the set consists of a set of corresponding, redundant devices (sensors and effectors). It is not required that these devices be interconnected by the same topology. To support function migration, each network in the set may have corresponding connections to more than one GPC. However, during normal operation, access to this set of networks is reserved exclusively for one GPC.

Redundant I/O Service: A special form of Local I/O Service where I/O activity is conducted on a set of Redundant I/O Networks. This is the only type of service supported on Redundant I/O networks. The intent of this service is threefold:

1) to provide simultaneous access to redundant devices on redundant networks during no fault conditions
2) to increase the bandwidth of the physical I/O networks communicating with redundant external devices.
3) to provide applications an uninterrupted flow of I/O data during periods of network reconfiguration activity.

Network Protocols

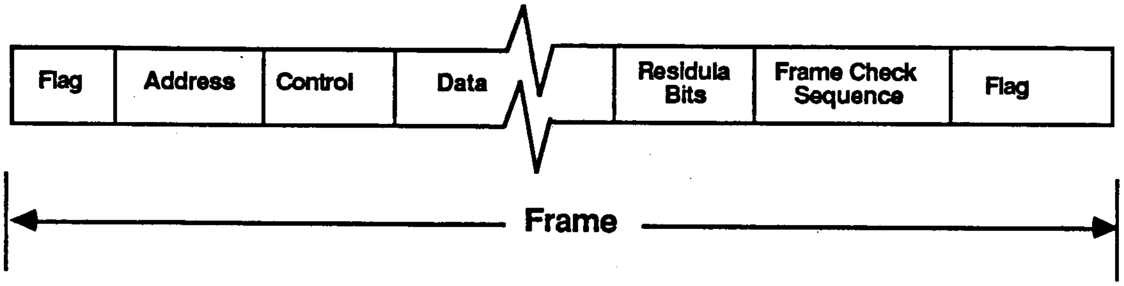
HDLC Protocol: The bit oriented protocol conducted on the data link of the communication hierarchy.

General I/O Protocol: The protocol followed between the IOS and nodes and DIUs for the purpose of conducting I/O transactions. All transactions begin with a command frame sent from the IOS to a node or DIU. A node always returns a response frame. A DIU is not required to return a response frame.

Network Traffic

Frame: An HDLC frame. The smallest unit of communication between an IOS and an external device (node or DIU) on an I/O network.

HDLC FRAME FORMAT



Flag Field:	01111110	Control Field:	1 byte	RB	0-7 bits
Address Field:	1 byte	Data Field:	1-122 bytes	FCS Field:	2 bytes

Command Frame: A frame sent to a single node or DIU from an IOS using the HDLC protocol. Sèe Figure ?? A command frame to a node is distinguished from a command frame to a DIU by the number of residual bits which are transmitted. A node command frame has three residual bits while a DIU command frame has zero residual bits.

Response Frame: A frame sent to a GPC from a node or DIU using the HDLC protocol. See Figure ?? A response frame from a node is distinguished from a response frame from a DIU by the number of residual bits which are transmitted. A node response frame has three residual bits while a DIU response frame has zero residual bits.

I/O Transaction: A command frame which may be followed by a response frame. A node always returns a response frame. A DIU is not required to return a response frame.

I/O Chain: An ordered set of one or more transactions addressed to devices on one I/O network. A chain consists exclusively of either node transactions or DIU transactions. A chain is the smallest unit of I/O activity conducted by an IOS for a GPC.

Redundant Chains : A set of I/O chains designed to execute in loose simultaneity on a set of redundant I/O networks. The transactions within each chain are in a one to one correspondence with the transactions in the other chains. This reflects the one to one correspondence of redundant DIUs among the networks.

I/O Request: A set of one or more I/O chains each of which executes on a different I/O network. An I/O request consists exclusively of one set of redundant chains or of one set of non-redundant chains. An I/O request is the smallest unit of I/O activity conducted by I/O System Services for a user.

Chain Execution: The activity carried out by an IOS which results in the transmission of command frames and the reception of response frames on an I/O network. The program which an IOS executes is under the direct control of I/O System Services and the indirect control of the user specifying the chain. When a user creates a chain of transactions, certain parameters must be specified which control the execution of the chain. I/O System Services then translates these specifications into a program which is stored in DPM and which executes when I/O System Services starts that chain. The activity is initiated by I/O System Services but executes independently of the program running in the GPC.

APPENDIX B: I/O SERVICE OPERATING RULES: NETWORK TOPOLOGY, GPC CONNECTIVITY AND I/O REQUEST DEFINITION

I/O Service: Definition and Operation

1.) An I/O Service provides access either to one regional network or to a set of one or more local networks.

2.) An I/O Service to a set of local networks operates those networks in parallel.

3.) I/O Requests are specific to one I/O Service. It consists of a set of chains, at most one per network within the service.

4.) All chains in an I/O Request are started at the same time. The I/O Request is completed, and data becomes available to a user, when all chains within the request are completed.

5.) Chains on parallel neworks can be used to allow corresponding devices on each network to be accessed at approximately the same time. The degree of simultaneity which can be achieved is determined by three factors: the rate at which the IOS samples its Interface Command Register, the amount of time required to issue a Start Chain command, and the reproducibility of the response time for corresponding external devices.

6.) A network is out of service from the time errors are detected on that network until a reconfiguration has been effected. In this context , a reconfiguration consists of either a network interface switch or a network reconfiguration. When a network is out of service, no user chains are executed on that network, however, service to other networks in that I/O Service is not affected.

7.) Node status collection and spare link testing will be conducted simultaneously on all parallel networks within an I/O Service.

Network Topology Rules

1.) Nodes will be connected in a way which would require at least 3 port failures to isolate any node or set of nodes from the rest of the network. This is the so called "minimum cut set ".

2.) At most one DIU will be connected to a node.

3.) At most one GPC will be connected to a node.

4.) A node may be connected to both a GPC and a DIU.

5.) Parallel networks need not be connected in identical ways nor do they need to contain the same number of nodes or the same number of DIUs. In this way, user can trade throughput for reliability.

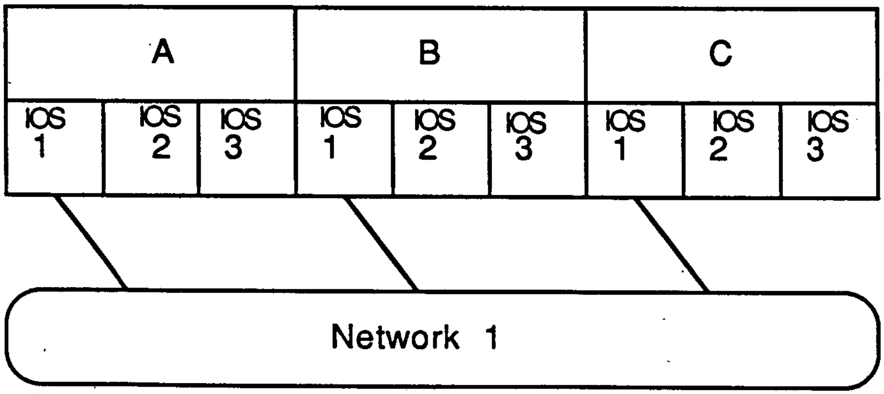
GPC Connectivity and Network Interfaces

1.) A network has at most one interface per GPC channel, i.e. redundant root links to a network from a GPC come from distinct channels. Thus the maximum number of network interfaces connecting a GPC to a network is equal to the number of channels in the GPC.

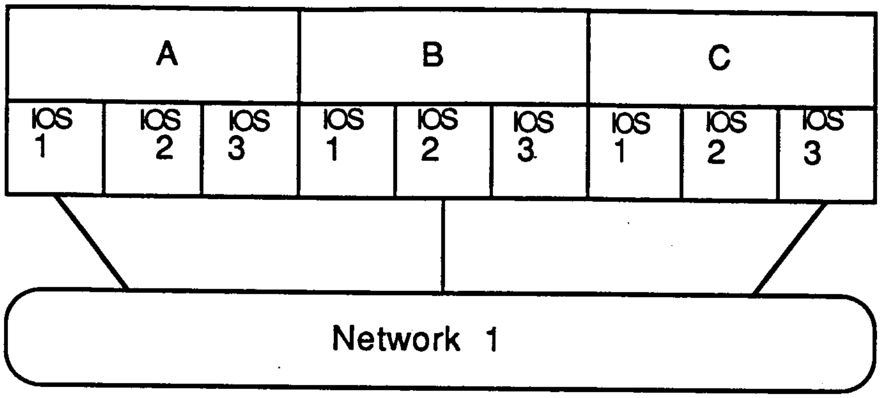
2.) Parallel networks are local networks in that they are used exclusively by one GPC for normal operations for long periods of time. However, more than one GPC may be physically connected to these networks and are therefore capable of taking over control and use of these networks in repsonse to failure conditions. These spare connections are made ready and initialized as if they were to be used but remain dormant until activated by some higher controlling process such as the system manager.

3.) Redundant network interfaces (i.e. root links to the same network) must have their IOSs occupy corresponding address spaces within their respective channels. This facilitates dual ported memory testing and allows modifications to chain programs and chain data to be made simultaneously to all redundant interface to the network.

CORRECT redundant root link connection of a GPC to a network



INCORRECT redundant root link connection of a GPC to a network



I/O Request Definition

1.) I/O Request Definitions determine whether an I/O Service is being used for reliability or throughput. They may access redundant devices simultaneously for greater reliability or they may access non-redundant devices for greater throughput.

2.) An I/O Request may run chains on a subset of networks in an I/O Service, however , unused networks in the service remain idle during the execution of the request.

APPENDIX C: INPUT OUTPUT SEQUENCER (IOS)

1.0 OVERVIEW

The Input Output Sequencer (IOS) is an autonomous asynchronous interface between an AIPS General Purpose Computer (GPC) and an I/O network. It resides on the shared bus of the GPC and can be accessed by either the Computational Processor (CP) or the I/O Processor (IOP). A major function of the IOS is to carry out detailed communication with I/O devices on the network as well as with the network nodes, off-loading the GPC from lower level I/O functions. A simplified block diagram of the AIPS I/O organization is shown in Figure 1.

The IOS is connected to a node of the I/O network via a bidirectional connection, which is called a root link. When activated by the GPC, the IOS can transmit on the network via its root link. The IOS is a memory mapped device that can be accessed and/or programmed by the CP or the IOP to perform a sequence of instructions which is called a chain. The memory locations within an IOS form a dual port memory that can be accessed by the processors or the IOS. GPCs preload data into this memory for transmission to Device Interface Units (DIUs) or nodes. The IOS interfaces with the DIUs and nodes in a command response mode, which is referred to as a transaction. During a transaction the IOS transmits a command to a DIU or node and then, if required, waits a predetermined time for a response. The IOS writes response data into the locations of memory specified by the GPC in the chain. An IOS executes a chain only when it is enabled by its GPC.

Each channel of a redundant GPC may contain an IOS. These IOSs, if connected to separate independent networks, can all be active simultaneously. However, if all of the IOSs are connected to the same I/O network, then only one should be enabled to transmit at a time. A GPC channel may also contain more than one IOS for redundancy. When an IOS is commanded to start, it first contends (polls) with all other active IOSs for the use of the network if that network is shared among several GPCs. If it wins, it then has exclusive use of the network and can send and receive messages to DIUs and nodes. If an IOS loses, it waits for the network to be quiet (no data traffic) for a fixed amount of time or for another poll to start before contending again. Provision is made within the IOS for starting a poll without waiting if a failure is perceived on the network.

A detailed explanation of the components of the IOS follows. It includes a description of the instruction format, memory assignments, register definition as well as chain examples for the IOS. For the purpose of this document a chain is defined as the instructions that are executed as a unit. All instructions within the IOS are 4 bytes long. All values are given in hexadecimal unless otherwise specified.

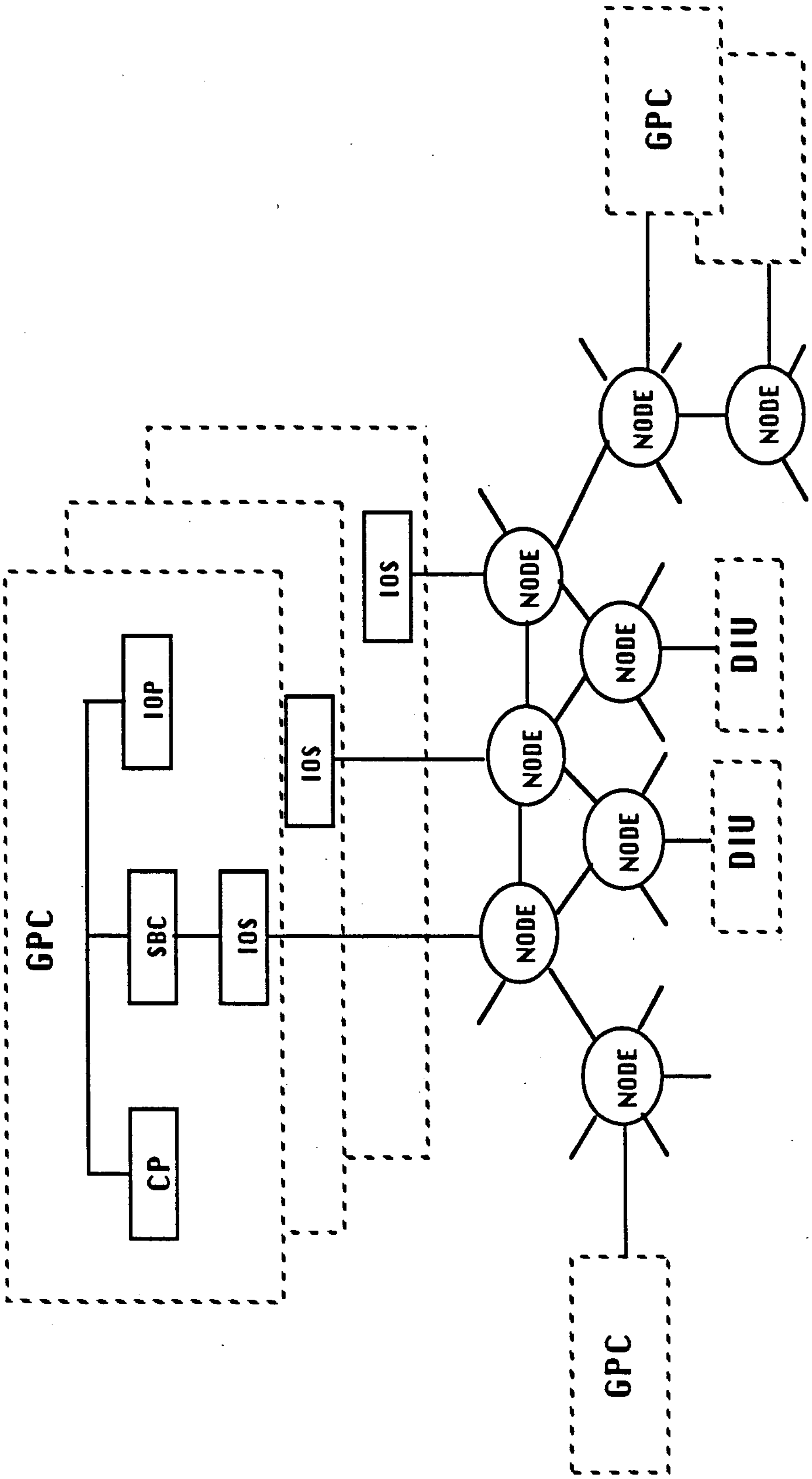


Figure 1. AIPS I/O Organization

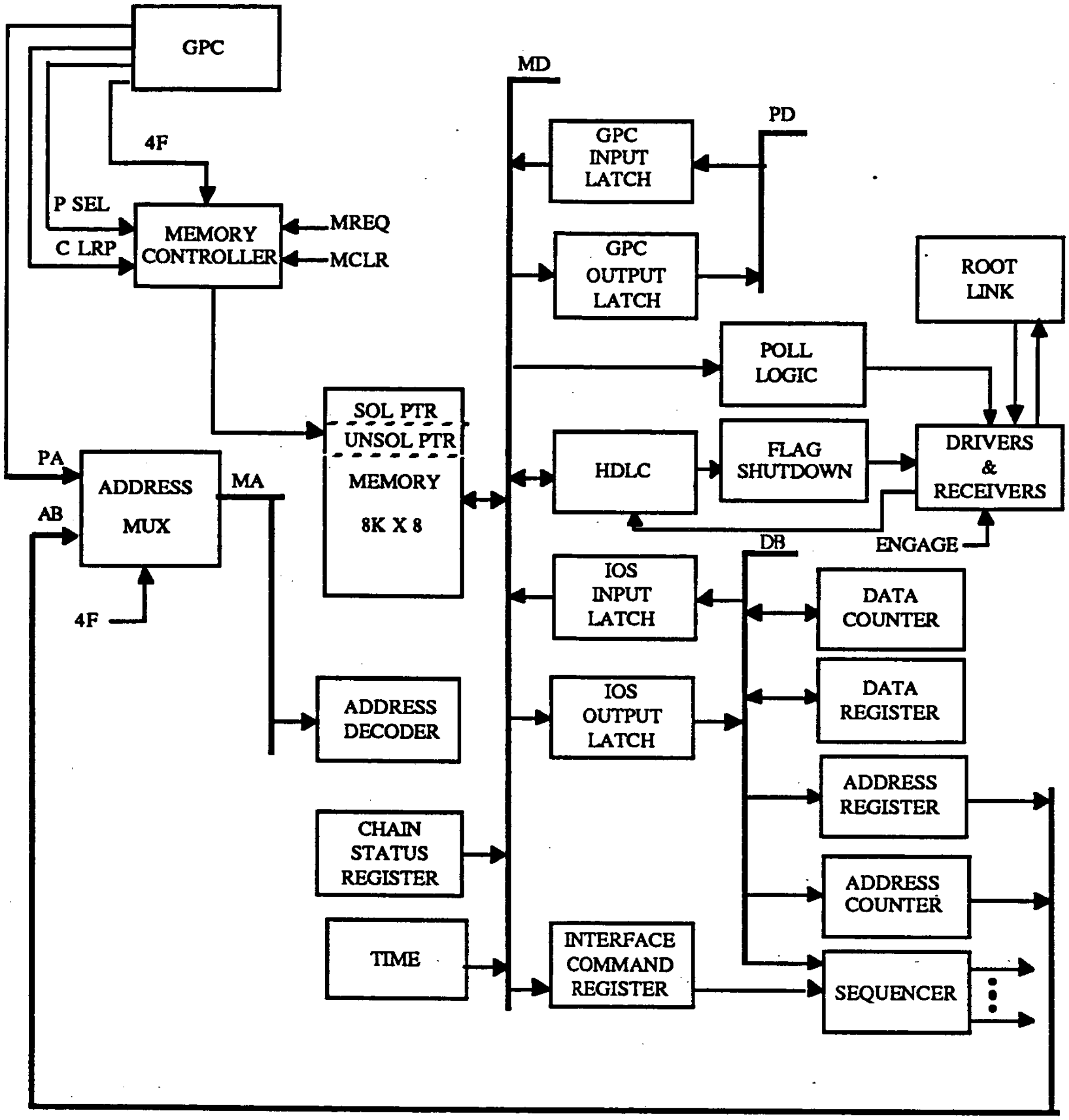


Figure 2. IOS Block Diagram

2.0 IOS ORGANIZATION

A block diagram of the IOS is shown in Figure 2. The IOS is programmed from a GPC which has access to the dual port memory and hardware registers. After loading the memory with the required chains the GPC then starts the IOS. The IOS can then poll and run the chains without GPC intervention. An overview of the major components of the IOS is given below.

2.1 MEMORY CONTROLLER — The memory controller arbitrates memory accesses from the GPC and the IOS. The memory is time shared between them by the use of processor signal 4F. When 4F is high the processor can access the memory and when 4F is low the IOS can have access. The memory controller generates chip select, read write and output enable at the appropriate times.

2.2 ADDRESS MULTIPLEXER — The address multiplexer selects between the GPC and IOS address buses. The output of the multiplexer is the memory address bus (MA). When 4F is high the processor address bus is connected to memory and when 4F is low the IOS memory bus is connected to the memory.

2.3 MEMORY — The IOS memory is a byte addressed memory containing 8192 bytes. It is used to store the chains, input packets and output packets. The first two bytes of memory are used as the solicited chain pointer and the second two bytes are used as the unsolicited chain pointer.

2.4 GPC INPUT LATCH — The GPC input latch is a buffer driver used to input byte wide data from the GPC data bus (PD) to the memory data bus (MD).

2.5 GPC OUTPUT LATCH — The GPC output latch is a buffer driver used to output data from the memory bus (MD) to the GPC data bus (PD).

2.6 IOS INPUT LATCH — The IOS input latch is a buffer driver used to input byte wide data from the internal IOS data bus (DB) to the memory data bus (MD).

2.7 IOS OUTPUT LATCH — The IOS output latch is a buffer driver used to output data from the memory bus (MD) to the internal IOS data bus (DB).

2.8 ADDRESS DECODER — The address decoder decodes the individual hardware registers which are located in the memory space between 10 and 1F. The addresses of the hardware registers is given in section 4.

2.9 INTERFACE COMMAND REGISTER — The interface command register is a write only register that contains the commanded mode. See section 5.3.1 for a detailed description of the possible modes.

2.10 SEQUENCER — The sequencer is the main control element of the IOS. When started, the sequencer fetches the instructions from memory, stores them internally, decodes and executes the microcycles by generating the appropriate control signals.

2.11 CHAIN STATUS REGISTER — The chain status register is a read only register that contains the chain and contention logic status within the IOS. See section 5.2.1.2 for a detailed description of the status reported.

2.12 ADDRESS COUNTER — The address counter stores the current memory address that the IOS is using. This address points to where the chain instructions are located in memory. During an input instruction it points to the location where the incoming data byte is to be stored. In an output instruction it points to the byte to be output when the HDLC chip requests a byte. It is loaded during instruction fetches and incremented during the instruction microcycles.

2.13 ADDRESS REGISTER — The address register contains the fixed addresses used in the instructions. During an input instruction it contains the address of the HDLC input register. During an output instruction it contains the address of the HDLC transmitter holding register.

2.14 DATA COUNTER — The data counter contains data that is incremented during an instruction. During an input instruction it accumulates the byte count of the incoming data. During an output instruction it counts the number of bytes outputted until the message is complete at which time it signals the sequencer to terminate the instruction.

2.15 DATA REGISTER — The data register is used to temporarily store data within an instruction. During an input instruction it holds the incoming byte from the HDLC receiver register until a memory cycle can be performed to store it. During an output instruction it holds the next byte to be outputted until the HDLC transmit holding register requests it.

2.16 HDLC — The HDLC device contains independent transmitter and receiver sections. The HDLC transmitter section receives the data bytes, appends opening and closing flags, encodes, and transmits the data. The receiver section searches the data stream for an opening flag. When it detects one, it synchronizes with the data fields and decodes the data stream into bytes for storage. In both modes the device generates the handshaking signals necessary to run the interface. See Sections 5 for details on the operation and control of the HDLC registers.

2.17 FLAG SHUTDOWN — The flag shutdown logic guarantees that the external IO network transmissions lines are always left in the same state after use. This allows the data and poll logic to utilize the same transmission lines. See section 6 for details.

2.18 DRIVERS and RECEIVERS — These drivers and receivers allow the IOS to interface to the IO network. The drivers are enabled by an engage line from the GPC. The receivers are always enabled but the input is controlled by the HDLC device.

2.19 POLL LOGIC — The poll logic allows the IOS to contend with other IOSs to gain control of the IO network. When enabled, the poll logic monitors the IO network waiting for a quiet time and then starts a poll. When it wins it starts a solicited chain, but if it loses it waits for the next poll or quiet time and tries again. See section 7 for additional details.

2.20 TIME DRIVER — The time driver allows the chain to read the time byte that appears on the shared bus.

3.0 INSTRUCTION FORMATS

The IOS can execute a limited number of instructions to perform its functions. The following paragraphs detail the form and function of the IOS instructions.

3.1 NOP (0000 0000) — This instruction updates the chain pointer to the address of the next sequential instruction. At the end of the NOP it will fetch that instruction.

3.2 BRANCH (2000 dddd) — This instruction will fetch the instruction contained at location 'dddd' and begin its execution. The Chain Pointer will be updated to point to the next instruction (dddd+4).

3.3 MOVE (40ss dddd) — This instruction will move a byte, located at any address 'ss' within the first 256_{10} bytes of IOS memory, to the byte address specified by 'dddd'. MOVE can be used to store the current value of a hardware register or store a preset value into a register.

3.4 MOVE IMMEDIATE (60xx dddd) — This instruction allows a constant, xx, to be stored into the destination address dddd.

3.5 INPUT (801B dddd) — This instruction will store incoming HDLC bytes in the buffer area starting at address 'dddd'. At the start of execution of this instruction the byte reserved for the input byte count is set to zero and the current value of the contention status is also stored within the buffer. As bytes are received they are stored at sequential addresses within the specified buffer locations and an internal byte count is incremented. A valid message always ends with a closing flag, which causes the IOS to then store the byte count, HDLC status registers and the TIME byte within the incoming packet buffer area. The INPUT instruction has now completed and the next sequential instruction is fetched and executed. The maximum number of data bytes that a single instruction can store is 122_{10}. If the INPUT contains more than 122_{10} data bytes, data will be lost. However, the buffer will never exceed the 128_{10} bytes allotted to it. The byte count which includes the status bytes, will also never exceed 80 (128_{10}). This instruction can be terminated if the time allotted for response is exceeded (the value programmed into the timer is reached without a data byte being received). However, in this situation none of the status information (HDLC IR & SR registers, time and byte count) is saved. An incoming data packet will always have the following format.

Byte count

HDLC IR register

HDLC SR register

TIME byte

contents of Chain Status Register

data (first byte)

•

•

data (last byte received)

3.6 OUTPUT (E01C ssss) — This instruction will transmit the bytes specified in the buffer starting at location 'ssss + 1'. The first byte at location 'ssss' contains the value of the expression, 80 - NB, where NB is the number of bytes to be transmitted. This instruction terminates when all the bytes have been transmitted.

4.0 MEMORY MAP

The following is the assigned memory locations in the dual port memory space of the IOS. Addresses 10 - 1F are hardware registers, however they are addressed the same as the RAM locations. All memory addresses, including the hardware registers are accessible from the CPU.

ADDRESS		FUNCTION
0	R/W	Solicited Chain Pointer - High Byte (RAM)
1	R/W	Solicited Chain Pointer - Low Byte (RAM)
2	R/W	Unsolicited Chain Pointer - High Byte (RAM)
3	R/W	Unsolicited Chain Pointer - Low Byte (RAM)
10	R	Chain Status Register
11	W	Interface Command Register
12	W	Timer Limit Register
13	W	Poll ID Register - 6 bit polling address

14	W	Poll Prio Register-3 bit prio & polling level
15	R	Time Byte
16		Reserved
17		Reserved
18	R/W	HDLC Control Register 1 (CR1)
19	R/W	HDLC Control Register 2 (CR2)
1A	R/W.	HDLC Control Register 3 (CR3)
1B	R	HDLC Receiver Holding Register (RHR)
1B	W	Address Register (AR)
1C	R	HDLC Interrupt Register (IR)
1C	W	Transmit Holding Register(THR)
1D	R	HDLC Status Register (SR)
1E		Reserved
1F		Reserved

With the exception of the addresses specified above, the rest of the dual port memory space can be used for any desired function. However, it should be noted that the MOVE instruction can only use the first 256_{10} addresses for the source byte.

5.0 REGISTERS

A description of the hardware registers and their use is contained in the following paragraphs. The hardware can execute two types of chains, solicited and unsolicited. Solicited chains are defined as command response chains and are meant to be executed when the GPC has control of the network. Unsolicited chains are defined as those that are performed when the GPC does not have control of the network but must accept all frames addressed to it. Unsolicited chains are not defined on the IO network, however, they are used as a vehicle while waiting for a poll to be won. On the IC network using the ICIS, unsolicited chains are executed whenever the GPC does not have the network, including while waiting for a poll to be won.

5.1 READ/WRITE REGISTERS

5.1.1 CHAIN POINTER REGISTERS

5.1.1.1 SOLICITED CHAIN POINTER (ADDR = 00 & 01) — The solicited chain pointer is used by the IOS to indicate where the next instruction of a solicited chain is located. When a new chain is to be started, this location is loaded with the address of the first instruction to be executed. It must be loaded before an execute chain command is issued. As each chain instruction is fetched, this location is updated to point to the next sequential instruction. The GPC can read this location at any time. However, since the IOS writes the locations a byte at a time and the GPC can read them as a word, the value read by the GPC may be incorrect if a chain is executing. The GPC should not attempt to write these bytes while a chain is executing, since it cannot be guaranteed that the IOS is not presently also modifying them.

5.1.1.2 UNSOLICITED CHAIN POINTER (ADDR = 02 & 03) — The unsolicited chain pointer is used by the IOS to indicate where the next instruction of an unsolicited chain is located. When a new chain is to be started, this location is loaded with the address of the first instruction of the unsolicited chain to be executed. It must be loaded before an execute chain command is issued. As each chain instruction is fetched, this location is updated to point to the next sequential instruction. The GPC can read this location at any time. However, since the IOS writes the locations a byte at a time and the GPC can read them as a word, the value read by the GPC may be incorrect if a chain is executing. The GPC should not attempt to write these bytes while a chain is executing, since it cannot be guaranteed that the IOS is not presently also modifying them. Unsolicited chains are identical to solicited chains and can execute any mix of instructions.

5.1.2 HDLC READ/WRITE REGISTERS

The following is extracted from the Western Digital data sheets on the HDLC chip (WD 1935). Definitions of bit polarity and sense have been modified to reflect what is seen by the AIPS system.

5.1.2.1 CONTROL REGISTER #1 (CR1) (ADDR = 18) — Control register 1 is used to specify the transmitter parameters and the transmitter and receiver enables. It can be loaded by a GPC or by a MOVE instruction in the chain.

<u>NOTE</u>: This register must always be loaded after CR2 and/or CR3. If CR2 and/or CR3 are ever changed, CR1 must again be reloaded after the change even if there are <u>no</u> changes being made to CR1.

7	6	5	4	3	2	1	0
ACT REC	ACT TRAN	TC 1	TC 0	TCL 1	TCL 0	DTR	MISC OUT

5.1.2.1.1 ACT REC (bit 7) — Activate receiver bit when set to a ZERO (0), the receiver is enabled to accept a data stream. When set to a ONE (1), the receiver will ignore any frames on the network.

5.1.2.1.2 ACT TRAN (bit 6) — Activate transmitter bit. When set to a ZERO (0), the encoder and transmitter are enabled to output data onto the network. When set to a ONE (1) the HDLC device will not transmit data.

5.1.2.1.3 TC1 and TC0 (bits 5 and 4) — The transmit command bits program the device into the requested mode. In AIPS, the OUTPUT instruction will function properly only in the data mode. These bits and the modes that they generate are as follows:

bit 5	bit 4	MODE	FUNCTION
1	1	data	Outputs the contents of the transmitter holding register
1	0	abort	Generates an abort message (not used on AIPS)
0	1	flag	Transmits one flag character (not used on AIPS)
0	0	FCS	Generates the two CRC bytes and a closing flag (not used on AIPS)

5.1.2.1.4 TCL1 and TCL0 (bits 3 and 2) — These bits control the number of bits per character from the transmitter. In AIPS this has been defined as 8 bit bytes. The definition of these bits follows:

bit 3	bit 2	BITS PER CHARACTER
1	1	8
1	0	7
0	1	6
0	0	5

5.1.2.1.5 DTR (bit 1) — Data Terminal Ready, a modem signal that is not used in this design and should be programmed to a ONE (1).

5.1.2.1.6 MISC OUT (bit 0) — Miscellaneous Output, a control signal not implemented in this design and should be programmed to a ONE (1).

5.1.2.2 CONTROL REGISTER #2 (CR2) (ADDR = 19) — Control register #2 specifies the receiver parameters and other control functions as defined below. It can be loaded by a GPC or by a MOVE instruction in the chain.

7	6	5	4	3	2	1	0
EXT CONT	ADDR COMP	EXT ADDR	RCL 1	RCL 0	LOOP	SELF TEST	AUTO FLAG

5.1.2.2.1 EXT CONT (bit 7) — This bit extends the HDLC control field. It is not used on AIPS and must be programmed to a ONE (1).

5.1.2.2.2 ADDR COMP (bit 6) — This bit enables the on-chip address comparator. If set to a ZERO (0), the first byte after the opening flag will be compared to the byte stored in the AR register. If equal, the data bytes that follow will be output. If address compare is enabled, and the address does not compare, all data bytes following will be ignored. If bit six is set to a ONE (1) then address comparison is not performed in the chip and all bytes between the opening and closing flag are presented to the interface. In AIPS, the IOS and the NODES do not use the address compare function but the ICIS does.

5.1.2.2.3 EXT ADDR (bit 5) — This bit extends the HDLC address field. It is not used on AIPS and must be programmed to a ONE (1).

5.1.2.2.4 RCL1 and RCL0 (bits 4 and 3) — These bits specify the receiver character length. In AIPS this has been defined as 8 bit characters. The definition of these bits is as follows:

bit 4	bit 3	BITS PER CHARACTER
1	1	8
1	0	7
0	1	6
0	0	5

5.1.2.2.5 LOOP (bit 2) — Specifies HDLC loop mode, a test function, not implemented in the IOS. This bit should always be programmed to a ONE (1).

5.1.2.2.6 SELF TEST (bit 1) — Chip diagnostic mode, not implemented through the IOS. This bit should always be programmed to a ONE (1).

5.1.2.2.7 AUTO FLAG (bit 0) — When this bit is set to a ZERO (0) and the transmitter is enabled, the chip will issue constant flag characters between frames. The IOS design utilizes this function and therefore must be set to a zero during an output instruction.

5.1.2.3 CONTROL REGISTER #3 (CR3) (ADDR = 1A) — This register is used to control the number of residual bits in a transmission. It can be loaded by a GPC or by a MOVE instruction in the chain. The definitions of these bits are as follows:

7	6	5	4	3	2	1	0
X	X	X	X	X	TRES 2	TRES 1	TRES 0

5.1.2.3.1 BITS 7 through 3 — Unused

5.1.2.3.2 TRES 2 - 0 (bits 2, 1 and 0) — These bits define the number of residual bits to be sent as the last character in a transmission. Messages sent to and from a NODE contain three (3) residual bits. Messages to and from DIUs contain no residual bits. The definition of these bits are as follows:

bit 2	bit 1	bit 0	RESIDUAL BITS/FRAME
1	1	1	No residual bits sent
1	1	0	1
1	0	1	2
1	0	0	3
0	1	1	4
0	1	0	5
0	0	1	6
0	0	0	7

5.2 READ ONLY REGISTERS

5.2.1.1 CHAIN STATUS REGISTER (ADDR = 10) — This register contains status of both the chain and the contention logic.

CHAIN STATUS REGISTER (Read Only)

7	6	5	4	3	2	1	0
Chain Comp	Contention State		Possession Default	Data Tx Fail	Poll Tx Fail	Any Rcv Fail	Any Rcv Good

5.2.1.1.1 CHAIN COMPLETE (bit 7) — This bit is set whenever the current chain has completed. Chain complete is defined as an IOS transition from solicited mode to unsolicited mode without the POLL bit in the command register set. It is reset whenever the poll bit is changed to a one in the interface command register or the IOS transitions from the unsolicited mode to the solicited mode.

5.2.1.1.2 CONTENTION STATE (bits 6 and 5) — This is the present state of the poll logic only. The following are the possible states that can be indicated.

5.2.1.1.2.1 INACTIVE, BUS RELEASED (00) — Both bits are zero whenever the IOS is not attempting to gain control of the network.

5.2.1.1.2.2 WAIT (01) — This IOS has been instructed to acquire the network, however no POLL has started since the request occurred.
5.2.1.1.2.3 ATTEMPTED (10) — This IOS has entered and lost at least one POLL sequence since being commanded to acquire the network.

5.2.1.1.2.4 POSSESSES (11) — This IOS presently has possession of the network.

5.2.1.1.3 POSSESSION DEFAULT (bit 4) — Indicates that the IOS possesses the network and detected an incoming POLL length bit on the network. If a chain is in progress when this happens, it will continue to completion. This bit is reset whenever the POLL bit in the Command Register is set to zero.

5.2.1.1.4 DATA TX FAIL (bit 3) — Indicates that a data bit was detected at the receiver during a command frame transmission. The chain will continue to completion. This bit is reset whenever the POLL bit in the Command Register is set to zero. This bit can only be set during a network possession.

5.2.1.1.5 POLL TX FAIL (bit 2) — Indicates that a data length bit was detected during a Poll Sequence. This bit is reset whenever the POLL bit in the Command Register is set to zero.

5.2.1.1.6 ANY RCV FAIL (bit 1) — Indicates that at least one response frame has been received with a protocol error in it. It is reset whenever a new poll begins or the IOS transitions from the unsolicited mode to the solicited mode.

5.2.1.1.7 ANY RCV GOOD (bit 0) — Indicates that at least one response frame has been received without a protocol error. It is reset whenever a new poll begins or the IOS transitions from the unsolicited mode to the solicited mode.

5.2.2 HDLC READ ONLY REGISTERS

5.2.2.1 RECEIVER HOLDING REGISTER (RHR) (ADDR = 1B) — This read-only register contains the received bytes as they are decoded from the frame. When executing an input instruction the IOS automatically reads this location and stores the received bytes into the specified location in the dual port memory.

5.2.2.2 INTERRUPT REGISTER (IR) (ADDR = 1C) — This read-only register contains status information on the state of the HDLC operation. It can be read by the GPC or with a MOVE instruction within a chain. Bits 7 through 3 will accumulate information such that if the IR is read after several operations, it will have the "OR" of all those frames. The definition of the bits within this register is as follows:

7	6	5	4	3	2	1	0
REOM	REOM	XMIT	XMIT	DISC	DRQI	DRQO	INTRQ
NO	WITH	NO	WITH				
ERR	ERR	ERR	URUN				

5.2.2.2.1 REOM NO ERR (bit 7) — When equal to a ZERO, this bit indicates that the frame was received without errors. If this bit is read before the closing flag is detected, it will not have been updated from the last frame.

5.2.2.2.2 REOM WITH ERR (bit 6) — When equal to a ZERO, this bit indicates that the frame was received with errors. If this bit is read before the closing flag is detected, it will not have been updated from the last frame. The errors that are reported here are: CRC, overrun, invalid frame and aborted frame.

5.2.2.2.3 XMIT NO ERR (bit 5) — When equal to ZERO, this bit indicates that the transmitted frame had completed without underrun errors.

5.2.2.2.4 XMIT WITH URUN (bit 4) — When equal to ZERO, this bit indicates that the transmitted frame had extra bytes inserted by the chip because the data was not available to the transmitter in the allotted time.

5.2.2.2.5 DISC (bit 3) — This bit is used with modems and in this system has no meaning.

5.2.2.2.6 DRQI (bit 2) — When set to a ZERO, this bit indicates that there is a byte available in the Receiver Holding Register (RHR). Reading the RHR sets this bit to a ONE. The hardware uses a buffered copy of this bit when storing bytes into dual port memory during an input instruction.

5.2.2.2.7 DRQO (bit 1) — When set to a ZERO, this bit indicates that the Transmit Holding Register (THR) is empty and requires another character to prevent an underrun error. Storing a byte into the THR sets this bit to a ONE. The hardware uses a buffered copy of this bit during an output instruction to read a byte from the dual port memory and store it into the THR.

5.2.2.2.8 INTRQ (bit 0) — This bit is set to a ZERO whenever at least one of bits 3-7 in the IR register is set to a ZERO. This bit is set to a ONE whenever the IR is read. A buffered copy of this bit is used to terminate a normally completing input or output instruction.

5.2.2.3 STATUS REGISTER (SR) (ADDR = 1D) — This read-only register contains status information that, when used in conjunction with the contents of the Interrupt Register, define the cause of the error.

7	6	5	4	3	2	1	0
RI	CD	DSR	MISC IN	RCVR IDLE	RRES 2 /ERR	RRES 1 /ERR	RRES 0 /ERR

5.2.2.3.1 RI (bit 7) — A modem signal not implemented in this interface.

5.2.2.3.2 CD (bit 6) — A modem signal not implemented in this interface.

5.2.2.3.3 DSR (bit 5) — A modem signal not implemented in this interface.

5.2.2.3.4 MISC IN (bit 4) — An input discrete not used in this interface.

5.2.2.3.5 RCVR IDLE (bit 3) — When set to a ZERO, the receiver is idle, i.e. a frame is not in process.

5.2.2.3.6 RRES2 /ERR (bit 2) — This bit has a dual role. If bit 7 in the Interrupt Register is a ZERO, then this bit is part of a binary number (see section 5.2.2.3.8) representing the number of residual bits received. If bit 6 in the Interrupt register is set to a ZERO, and this bit is set to ZERO then an aborted or invalid frame was detected.

5.2.2.3.7 RRES1 /ERR (bit 1) — This bit has a dual role. If bit 7 in the Interrupt Register is a ZERO, then this bit is part of a binary number (see section 5.2.2.3.8) representing the number of residual bits received. If bit 6 in the Interrupt register is set to a ZERO, and this bit is set to ZERO then an overrun error was detected. An overrun error indicates that a received byte was not removed from the Receiver Holding Register before the next byte was received. That first byte will be lost.

5.2.2.3.8 RRES0 /ERR (bit 0) — This bit has a dual role. If bit 7 in the Interrupt Register is a ZERO, then this bit is part of a binary number (see below) representing the number of residual bits received. If bit 6 in the Interrupt register is set to a ZERO and this bit is set to ZERO, then a CRC error was detected.

bit	bit	bit	RESIDUAL BITS/FRAME
2	1	0	
1	1	1	No residual bits sent
1	1	0	1
1	0	1	2
1	0	0	3
0	1	1	4
0	1	0	5
0	0	1	6
0	0	0	7

5.2.3 TIME (read only) (ADDR = 15) — This byte contains a value that is slaved to the system timer, incremented by a 66_{10} microsecond clock and capable of measuring 16.830_{10} milliseconds. It can be read by the GPC or by a move instruction in the chain. It is automatically appended to all incoming frames that complete in a valid manner.

5.3 WRITE ONLY REGISTERS

5.3.1 INTERFACE COMMAND REGISTER (Write Only) (ADDR = 11) — This register contains the necessary control bits to operate the IOS. The following are valid commands used to control the IOS. The END CHAIN command transitions the IOS from solicited to unsolicited mode. The STOP CHAIN command turns the IOS off.

START CHAIN WITH POLL = 94

START CHAIN WITHOUT POLL = 80

END CHAIN = 84

STOP CHAIN = 20 (followed by a 00 command to prime the interface for the next command)

INTERFACE COMMAND REGISTER (Write Only)

7	6	5	4	3	2	1	0
EXECUTE CHAIN	X	STOP IMM	POLL	SPOLL	EXECUTE UNSOL CHAIN	X	X

5.3.1.1 EXECUTE CHAIN (bit 7) — When only the execute chain bit is set to a one (1), this commands the hardware to fetch and start executing instructions starting at the address stored in the Solicited Chain Pointer. The chain will start even if a Poll was neither started nor won. If however, a poll is to be won first before starting the chain, then bits 7, 4 and 2 must be set to a one. The hardware will then start the polling logic, start an unsolicited chain pointed to by the unsolicited chain pointer (usually an input instruction) and when a poll is won, automatically start the chain at the location pointed to by the solicited chain pointer.

5.3.1.2 Bit 6 - Not used by the IOS.

5.3.1.3 STOP IMM (bit 5) — When the stop immediately bit is set to a one (1) the hardware will turn off the IOS. Whatever function the IOS is now performing will be terminated. This allows the GPC to stop the hardware if it is caught in a loop or otherwise malfunctioning.

5.3.1.4 POLL (bit 4) — Whenever the poll bit is set to a one (1) the logic will attempt to gain control of the network by joining the next possible poll sequence. At the end of a chain this bit must be reset.

5.3.1.5 SPOLL (bit 3) — Whenever the spoll bit is set to a one (1), the hardware will immediately start to poll. The hardware will not wait for the start of a new poll from another site or an idle condition on the network. At the end of a chain this bit must be reset.

5.3.1.6 EXECUTE UNSOL CHAIN (bit 2) — This bit is only recognized by the hardware when set in conjunction with the execute chain bit, bit 7. If bits 7 and 2 are both set to a one (1), the hardware will execute the chain starting at the location pointed to by the unsolicited chain pointer. If a GPC desires to first gain control of a network, it sets bits 7, 4 and 2 to a one and all others to a zero (0). The hardware will enable the polling logic, start the unsolicited chain at the location pointed to by the unsolicited chain pointer (usually an input instruction) and when a poll is won, automatically start the chain at the location pointed to by the solicited chain pointer.

5.3.1.7 Bits 1 and 0 are not used.

5.3.2 HDLC WRITE ONLY REGISTERS

5.3.2.1 ADDRESS REGISTER (AR) (ADDR =1B) — This write-only register contains the address that the chip is to use for comparison if on-chip address recognition is being used. If on-chip address detection is not used, the contents of this register will be ignored. This register is not used by the IOS.

5.3.2.2 TRANSMIT HOLDING REGISTER (THR) (ADDR = 1C) — This write-only register holds the next data byte to be transmitted. The hardware loads a byte into this register during an Output instruction whenever DRQO is set.

5.3.3 TIMER LIMIT REGISTER (Write only) (ADDR = 12) — The timer limit register contains the current value to be used to time out an instruction. A non-zero value written to the timer limit register allows the timer to function. The timer is initialized at the beginning of each instruction and as each incoming data byte is detected. If an instruction does not complete or an incoming data byte is not detected in the programmed number of microseconds, the current instruction is terminated and the next sequential instruction started. A new value stored in the timer limit register will be utilized when the next instruction is started or the next incoming byte is detected during an input instruction.

5.3.3.1 TIMER LIMIT VALUE — The timer limit is the number of periods of the clock 2F. 2F has a period of approximately 2 microseconds. The timer, therefore, has an approximate range of 2 to 512 microseconds.

5.3.4 POLL REGISTERS

5.3.4.1 POLL PRIORITY REGISTER (Write only) (ADDR = 14) — The Poll Priority Register contains the six high order polling bits. The three bits labeled PRIO, are used for the initial priority of this IOS. They will automatically increment after each poll sequence loss until they contain all ones, at which time incrementing is inhibited and the maximum priority held. Since the initial state of the PRIO bits are not saved, this register must be reloaded whenever the initial polling state is required. The three bits labeled LEVEL, are the high order bits of the poll sequence. For the IO network LEVEL 2 is set to a one, LEVEL 1 and LEVEL 0 are set to a zero. It can be loaded by a GPC or by a MOVE instruction in the chain.

7	6	5	4	3	2	1	0
X	LEVEL	LEVEL	LEVEL	X	PRIO	PRIO	PRIO
	2	1	0		2	1	0

5.3.4.2 POLL ID REGISTER (Write only) (ADDR = 13) — The Poll ID register contains the six (6) low order bits used in the polling procedure. These bits normally contain the address that this IOS uses for polling. It can be written into by the GPC or by a MOVE instruction within the chain.

7	6	5	4	3	2	1	0
X	X	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
		(MSB)					(LSB)

6.0 FLAG SHUTOFF SYNC

The IOS uses the same IO network lines to communicate and to poll. In order to be able to perform both functions on the same lines all operations must leave the lines in a known state. The HDLC protocol allows the signalling lines to be left in either state, and in fact the device used to generate the HDLC protocol does leave the line in either state depending upon the data content of the message. The IOS contains logic, which upon sensing the end of a message, utilizes the closing flags to turn off with the line in a low state without generating any extraneous edges. When the next output message is started, the first flags are used to turn the logic back on to the state that the HDLC device attempted to leave the line. Again this is done without generating any extraneous bits. The polling logic is fabricated so as to always end with the line low.

7.0 POLLING

The IOS contains logic which allows it to contend with the other IOSs for use of the IO network. If the IOS is to contend for the network, the bits in the interface command register must be set to execute, poll and execute unsolicited mode. The logic will start the chain pointed to by the unsolicited pointer and simultaneously prime the polling logic. The reason for having a unsolicited chain is to give the IOS a place to wait for the poll to complete. Therefore, there must be an input instruction without the timer running where the IOS will "hang" waiting for the poll to be won.

The polling logic waits for either a poll to begin or the bus going quiet for 512 microseconds. When either occurs, the logic will assert a start bit for 24 microseconds. This gives all other IOSs time to recognize the start of a poll and join if required. At the end of the poll bit the logic compares the state of its input line with the state of its output line. If another IOS is joining the poll, the input line will also be high and the IOS must continue to poll to determine who will win. It next asserts the fixed priority bits, one at a time for 24 microseconds, followed by the variable priority bits and its address bits. At the end of each 24 microsecond period it compares its output to what it perceives on the bus. If what it hears is the same as what it is transmitting it must continue to the next bit as no decision can be made. If it hears a zero while it is transmitting a one, then it knows it has won because it has a higher value than all others that are contending. If it hears a one while it is transmitting a zero, then it knows it has lost because it has a lower value than at least one other contender, and it will stop transmitting and wait for another poll to begin. When the IOS decides that it has won it will abort the unsolicited chain and perform a context switch to the solicited chain and start to execute it.

The variable priority bits are incremented after each poll sequence loss until they reach the maximum value of 7. They will remain at this value until written into by the program or the chain. If an IOS detects a data bit during its polling it will terminate the poll and set an error bit.

8.0 DUAL-PORT OPERATION

The IOS utilizes a time shared 8k x 8 memory for program and input output buffer storage. This memory can be alternately accessed by the GPC and the IOS. Each site has independent access to the memory for four CPU clock periods.

8.1 TIMING The dual-port memory utilizes the CPU clock signal 4F, which has a period of eight CPU clocks. When 4F is high the GPC has access to the memory and when 4F is low the IOS has access to the memory. In the following discussion the IOS timing is discussed. The GPC timing is identical, happening on the opposite phase of 4F.

If the IOS requires the use of the memory it assert the signal MREQ. MREQ is recognized on the first rising edge of CPU clock after 4F falls, which causes a chip select to the memory to be asserted. (4F changes state on a falling edge of CPU clock.) Chip select is three clock periods wide. The memory cycle is terminated by MCLR being asserted for one CPU clock period and chip select being deasserted. MCLR causes MREQ to be deasserted.

When a write is specified, the read/write line will be low. One clock period after chip select is asserted, a write enable signal to the memory is asserted. If a read is specified, the read/write line will be high and on the next rising edge of CPU clock after chip select is asserted, an output enable will be asserted. By delaying output enable, none of the memory switching transients are seen.

The operation of the dual-port memory from the GPC side is identical except the memory request is initiated with the falling edge of PSEL and terminated with the assertion of CLRP. The GPC can only make memory requests during the time that 4F is high. The address multiplexers are also driven by 4F.

9.0 HDLC PROTOCOLS

The HDLC bit orientated protocol was chosen for use on AIPS. HDLC allows automatic address detection, control information and a cyclic redundancy error word to detect transmission errors. In the IOS automatic address detection and the control byte are not used. The IOS operates in a command response mode at all times. It sends a message to a site and then waits for a response only when it has control of the IO network.

An HDLC frame contains an opening flag, address byte, control byte, data bytes (in AIPS up to 119_{10}), FCS byte, FCS byte and a closing flag. The opening and closing flag are identical and consist of a zero, followed by six ones and a zero. It is not possible for a flag to look like data since the HDLC protocol specifies that within the data field after five continuous ones a zero is added.

10.0 ENGAGE

The AIPS GPCs generate a voted engage signal which is used to enable external functions. In a faulty GPC this signal will not be asserted. The IOS uses this signal to enable its bus driver that connects it to the IO network. Therefore, a faulty GPC and/or faulty IOS can be disconnected and prevented from bring down the IO network.

11.0 BUFFER FORMATS

A typical chain will contain both input and output instructions. Each of these instructions must have buffer areas within the IOS's memory. The input buffers contain the messages that the IOS receives from Nodes and DIUs. The output buffer areas contain the messages that the IOS sends to Nodes and DIUs. There are no restrictions on where in memory inputs or output are stored. The following is the format of the input and output messages.

11.1 INPUT BUFFER FORMAT: The third and fourth byte of the input instruction point to a location in memory where the IOS will store an incoming message. Each incoming message contains a five byte preamble before the data part of the message. The first byte contains the byte count, which is the number bytes received plus the four additional bytes of the preamble. This can be used as an offset to point to the last byte of the buffer. If the input instruction is terminated by the timer expiring, then this byte will contain zero even if a partial message had been received before the message stopped. The second and third bytes contains the HDLC IR and SR registers respectively. These bytes are used to check for HDLC protocol errors. The fourth byte contains a time tag as recorded at the end of the input instruction. The fifth byte contains the contents of the Chain Status Register. From the sixth byte on is the data content of the message. To recap, input buffers within the memory all have the following format:

Byte Count

HDLC IR Register

HDLC SR Register

Time

Content of Chain Status Register

data (first byte)

•

•

•

data (last byte)

In the case of a response from a Node the input format will be as follows:

Byte Count

HDLC IR Register

HDLC SR Register

Time

Content of Chain Status Register

Node Address

Port Activity Seen

Transmission Errors Seen

Valid Frame Seen

Error in Node Messages Seen

Node Port Configuration

Sum Check

Residue Bits (3 bits residue + 5 bits FCS)

FCS (next 8 bits of FCS)

FCS (last 5 bits of FCS + 3 bits of pad)

11.2 OUTPUT BUFFER FORMAT: The third and fourth bytes of the output instruction contain the address within the IOS memory of the output buffer for this instruction. The first byte located at this address is 80 - NB, where NB is the number of bytes in this output message. Following the byte count is the rest of the message. Since the longest message that can be received has been defined as 128_{10} bytes, and each input message contains a 5 byte preamble and 2 FCS bytes, the maximum data part of an output message can only contain 121_{10} bytes. If more than 121_{10} bytes are specified, the receiving location will truncate the message. The format of the output buffer is as follows:

Byte Count (80 - NB)

data

•

•

•

last data byte

12.0 EXAMPLE CHAINS

The following are intended to show how a Chain is programmed in the IOS.

12.1 EXAMPLE #1 — This example shows a chain which programs the HDLC chip and then does an Output frame followed by an input frame. The GPC stores the following values into the IOSs memory. (The IOS is a byte oriented device. For simplicity, the columns value or contents are two bytes and the columns labled IOS location or address show the address of the high order byte. i.e. 0100 @ 8CX000 means that a 01 is stored at location 8CX000 and a 00 is stored at location 8CX001. The value of X indicates in which channel of a GPC the IOS is located. i.e. X = 1 for channel A, X = 2 for channel B, X = 4 for channel C. The high order bit of X is the high order bit of the address of the dual-port memory.)

The GPC writes the following locations:

xx	8CX013	Value of low order polling bits
4y	8CX014	Value of high order polling bits
94	8CX011	Commands IOS to execute chain, execute unsolicited and poll

The last store writes into the interface command register which instructs the IOS to enter a POLL as soon as it detects one starting, or to start a POLL if it sees the bus go idle. The IOS meanwhile starts to execute the unsolicited instructions starting at location 200. As soon as this IOS thinks it won a POLL, it terminates the unsolicited chain and starts the solicited chain at location 100. (All values below are given in HEX.)

INST #	ADDRESS	CONTENTS	DESCRIPTION
--	8CX000	0100	Solicited Chain Pointer
--	8CX002	0200	Unsolicited Chain Pointer
	•	•	
	•	•	
0005	8CX100	4015	MOVE the current value of

	8CX102	01F1	time to location 01F1 (This could be done to find out when the solicited part of the chain started)
0006	8CX104 8CX106	401C 01F2	Read the IR register to clear any prior status
0007	8CX108 8CX10A	60FC 001A	Store an FC in CR3. Sets the chip to send 3 residual bits.
0008	8CX10C 8CX10E	60FE 0019	Store an FE in CR2. Puts the chip in the auto flag mode. This is mandatory to guarantee that all receivers will see the flag character and no extraneous data.
0009	8CX110 8CX112	60BF 0018	Store a BF in CR1. This enables the chip in the data mode and turns on the transmitter. (CR1 must be loaded after CR2 and CR3) Flags will now be sent until data is loaded into the THR.
0010	8CX114 8CX116	E01C 1000	Enter the OUTPUT mode. The byte count is read from location 1000 and the data bytes starting at location 1001 are transmitted. When the byte count reaches 80 the output instruction ends.
0011	8CX11C 8CX11E	607F 0018	Store a 7F in CR1. This instruction turns off the transmitter and turns on the receiver.
0012	8CX120 8CX122	401C 01F3	Read the IR register and store it in location 01F3. This will clear the status before the next use of the HDLC chip.
0013	8CX124 8CX126	401D 01F4	Read the SR register and store it in location 01F4.

0014	8CX128 8CX12A	60FF 0012	Store a FF in the timer limit register and enable it to run (Timer = 512 micro).
0015	8CX12C 8CX12E	801B 4000	Enter the INPUT mode. Location 4000 will be cleared to accept the incoming byte count. If no data is received the IOS will wait here for 512 microsecond before going on to instruction #17. If any data byte is received before a timeout, the timer will be restarted. The IOS will stay in this instruction until a closing flag is received or the timer expires or in the case of an infinite input message the GPC terminates the chain.
0016	8CX130 8CX132	6000 0012	Disable the timer.
0017	8CX134 8CX136	2000 0348	BRANCH to the next instruction to be executed in this chain at location 0348. (This is an example of how bypassing might be done. The next executable instruction will be at location 0348).
0001	8CX200 8CX202	4015 01F0	MOVE the current value of time to location 01F0. (This could be done to log the time that this device was first enabled.)
0002	8CX204 8CX206	6000 0000	Turn off the timer.
0003	8CX208 8CX20A	801B 1F00	Enter the INPUT mode and store the frame starting at location 1F00. In an IOS, the system will hang at this instruction for a poll

			to be won since there are no unsolicited messages on the I/O network.
0004	8CX20C	2000	In an IOS there would be only these two instructions. This BRANCH allows the IOS to return to unsolicited mode without the need to restore pointers.
	8CX20E	0208	

A possible way that a solicited chain could always end is the following. The last instruction in the chain does a branch to a location that performs the desired chain termination. The advantage of this is that the solicited chain pointer will always have a known value in it whenever a chain has gone to completion.

nnnn	xxxx	2000	This is the last instruction of the chain.
	xxxx+2	0FF0	It specifies BRANCH to 0FF0.
	8CXFF0	6084	This is an END CHAIN command.
	8CXFF2	0011	It turns off the POLL and places the IOS in the execute unsolicited mode. The solicited chain pointer will contain the value 0FF4, which can be used to verify that the chain has completed.

APPENDIX D: NODE SPECIFICATION

The input output network is comprised of simplex nodes. Nodes are interconnected by links. A node is a communication switching point with five input/output ports. Figure 1 is a basic representation of a node. The internal construction of each port of a node is shown in Figure 2. Since a node does not have knowledge of the configuration of the network it must always have its receivers enabled. Reconfiguration commands can be accepted from any port whether enabled or not. Configuration commands enable selected ports. Ports are reconfigured whenever necessary and can be temporarily modified for single response frames. As a message is received on an enabled port it regenerates and retransmits the received data. At the same time, the message is decoded within the node. If the message is addressed to the node it responds to the command embedded within the data. If the message is addressed elsewhere it checks for a valid transmission, latches observed error conditions and resets the receiver for the next transmission.

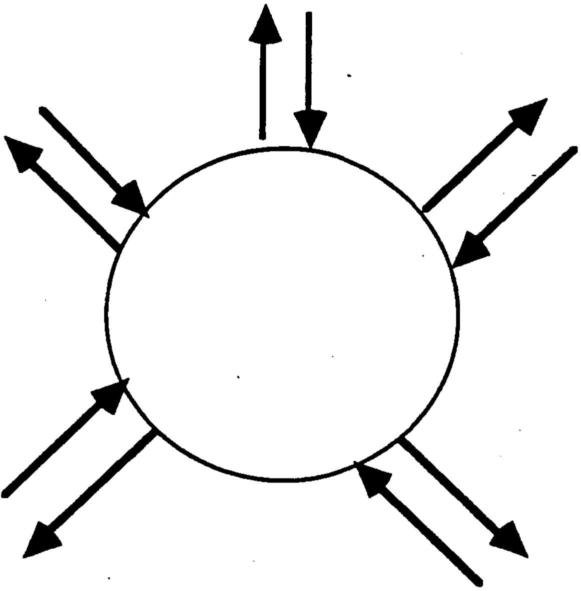


Figure 1. AIPS NODE

Some components are unique to a port and some are shared by all the ports. Figure 2 shows the basic construction of a node. The components within the dotted lines are unique to each port and are repeated five times. The components outside of the dotted lines form the node control section and are not repeated. The following is a description of the basic components of the node.

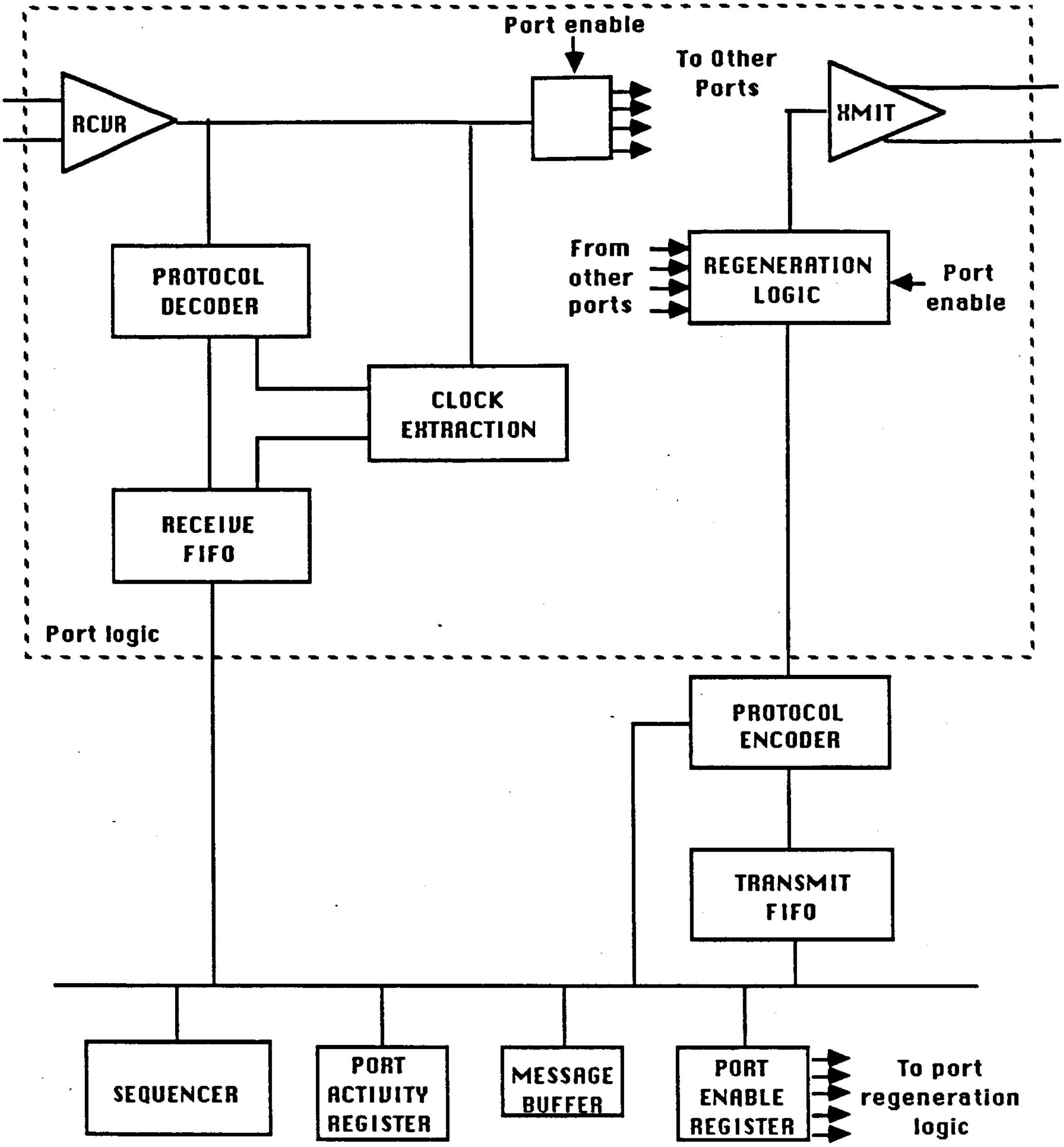


Figure 2. NODE PORT

Port Components (unique to each port)

1. Receiver

The receiver accepts the signal level on a link and converts it to the internal logic level of the port. The receiver also isolates the node from electrical failures of the link.

2. Protocol Decoder

The protocol decoder accepts the serial data stream from the receiver, checks for protocol compliance and transmission induced errors. It then assembles the message into parallel words utilizing its clock extraction section. These parallel words are stored in a receive fifo for the control sequencer to examine.

3. Clock Extractor

Since the data transmission rate is 2 MHz, and all elements (GPC's, nodes,etc.) are operating on independent oscillators, it is necessary to generate a clock for the decoder. This clock is synchronized to the first edge of data that it sees and remains usable for the maximum message length.

4. Signal Regeneration Logic

The signal regeneration logic is used to reconstruct the fidelity of the transmission. The passage of the signal through circuit elements in the node and the variability of the frequency of individual oscillators would degrade the signal if it were not reconstructed in each node. After several transitions through circuit elements the transmission could appear to be modified. The input to the regeneration logic is the OR of all the enabled port receivers and the protocol encoder output. The output of the regeneration logic is enabled or disabled by the port enable register and is applied to the input of the port transmitter.

5. Transmitter

The transmitter converts the output of the regeneration logic into the signaling levels used on the links.

Control Components (shared by all the ports)

1. Node Sequencer and Control

The node sequencer and control orchestrates the total operation of the node. It scans the port receive fifo's for messages received from the links. If a message is found, it checks the address byte to determine if the message is addressed to this node. If it is, it then checks the bytes that follow the address byte to see if the rest of the message conforms to a proper node message format. The message is acted upon only if it passes all tests. The sequencer is capable of reading the input fifo's, writing to the transmitter fifo, port enable register and message buffer.

2. Port Enable Register

The port enable register accepts the decoded commands from the sequencer and is used to enable and disable the individual port reconstruction logic. The last command is stored until rewritten by the next command. The contents of this register is contained within the status message from the node.

3. Message Buffer

The message buffer is a 64 byte long RAM which can be written into by an appropriate node command. The contents of this RAM can be returned by the Node in place of a status message.

4. Port Activity Register

The port activity register is set whenever a transition is detected on the port receiver.

5. Transmit Fifo

The transmit fifo holds the node response message for application to the protocol encoder.

6. Protocol Encoder

The protocol encoder receives the node responses and encodes them into the link protocol. The output of the encoder is sent to the reconstruction logic of all the ports.

Input Frame Message Format

The following is the format of an input frame sent to a node

1. Opening Flag
2. Node Address
3. Encoded Node Address
4. Operation Code
5. Port Enables and Control
6. Message Sum Check
7. Residue Bits
8. FCS
9. FCS
10. Closing Flag

Bit assignments within the transaction are as follows.

bit	7	6	5	4	3	2	1	0
Opening Flag	0	1	1	1	1	1	1	0
Node Address	Node Address Bits							
Encoded Address	Encoded Node Address Bits							
Op Code	Mode	Mode	Mode	Stat MsgB	Err	Res	Res	Res
Port Enable	Chg Port	Enb Once	Clr Stat	E	D	C	B	A
Sum Check	Sum Check Bits							
Residue Bits	Residue Bits							
FCS	FCS High Byte							
FCS	FCS Low Byte							
Closing Flag	0	1	1	1	1	1	1	0

1. OPENING FLAG: As defined in the HDLC specification, used to synchronize and separate transmissions. Recognized and extracted by the HDLC device.

2. NODE ADDRESS: The address of the Node to which this message is directed.

3. ENCODED NODE ADDRESS: The encoded address of the Node to which this transaction is directed. It has been placed in the byte that HDLC has defined as control. Since control code definition is defined by the user, in AIPS it will be used as the encoded address to help shorten the response time and is the 1's complement of the node address.

4. OPERATION CODE: Contains the code for the function to be performed by the addressed node. The following is the definition of those functions.

Bit	7	6	5	4	3	
	1	1	1	R	E	Modify Port Enable Register as specified in next byte.
	1	1	0	R	E	Reserved
	1	0	1	R	E	Next byte to count register
	1	0	0	R	E	Next byte to Address Reg H
	0	1	1	R	E	Next byte to Address Reg L
	0	1	0	R	E	Next byte to address specified by Address Register
	0	0	1	R	E	Next byte to address specified by Address Register then +1 to Address Register
	0	0	0	R	E	No modification to Port Enable Register (next byte ignored).

All valid input frames result in a response frame from the Node. The content of the response frame is determined by the state of bit 4 as defined below.

Bit 4 R=1 Respond from Status Register
R=0 Respond from Message Buffer

The node can be commanded to send a response frame that contains a transmission error for testing purposes. This faulted frame can occur in conjunction with any of the above defined modes. A faulted frame is one in which the transmission is truncated, i.e. aborted. The choice of valid or faulted frames is determined by the state of bit 3 as defined below.

Bit 3 E=1 Respond with faulty Message
E=0 Respond with valid message

Modes 1, 2, 3, 4 and 5 are for specifying the parameters used to generate responses from the message buffer. If a response is specified from the message buffer, the node will respond with the number of bytes specified by the counter starting at the address contained in the Address Register. The contents of the counter and Address Register are not changed by a response request. The counter and Address Register are modified as specified above using modes 3, 4 and 5. Modes 1 and 2 are used to load specified memory locations within the Node. Mode 1, when specified, automatically increments the address register after each byte is stored at the present location specified by the address register. The Address register can only specify locations from 00C0 H to 00FF H, a total of 64 bytes. Mode 2 is used to specify memory locations in a random access mode. Bits 2, 1, and 0 specify, in binary, the number of residue bits to be generated in a response frame.

5. PORT ENABLES AND CONTROL: If mode 7 is specified, in the opcode byte, this byte is loaded into the port Enable register. If bit 7, of this byte, is set (=1) then the port enable register is changed permanently. However, if bit 7 is not set and bit 6 is set, the contents of the port enable register are modified for this transmission only. At the completion of this transmission the previous contents are reloaded into the port enable register. If both bits 7 and 6 are set at the same time, the node will respond as if only bit 7 were set, i.e. the port enable register will be permanently modified. Bit 5, if set, specifies that all status registers are to be cleared after this response is completed.

6. MESSAGE SUM CHECK: The contents of this byte is calculated such that an add, modulo 256,of the Address byte, Encoded Address byte, OpCode byte, Port Enable byte and this byte will yield a result of zero. It is computed at the source and verified in the Node to check for errors outside the transmission medium.

7. RESIDUE BITS: Used to differentiate Node messages from all other transactions. There are three residue bits in a node message and the content of these bits is not specified.

8. FCS: This byte contains the high byte of the FCS as calculated in the transmitter.

9. FCS: This byte contains the low byte of the FCS as calculated in the transmitter.

10. CLOSING FLAG: This byte is defined by HDLC as the transmission terminator or separator. Detected and extracted by the HDLC device.

Output Frame Message Format

The node always responds after a valid input frame. The output frame can be generated from either the status register or the message buffer.

Output Frame From Status Register

When an output frame is requested from the status register it will take the following form.

1. Opening Flag
2. Node Address
3. Port Activity Seen
4. Transmission Errors Seen
5. Valid Frame Seen
6. Error in Node Messages Seen
7. Node Valid Frame Seen
8. Node Port Configuration
9. Sum Check
10. Residue Bits
11. FCS
12. FCS
13. Closing Flag

Bit assignments within the Output Frame from the status register are as follows.

bit	7	6	5	4	3	2	1	0
Opening Flag	0	1	1	1	1	1	1	0
Node Address	Node Address Bits							
Activity Seen	X	X	X	E	D	C	B	A
Transmission Errs	X	X	X	E	D	C	B	A
Valid Frame Seen	X	X	X	E	D	C	B	A
Node Error Seen	X	X	X	E	D	C	B	A
Node Valid Frame	X	X	X	E	D	C	B	A
Node Port Config	X	X	X	E	D	C	B	A
Sum Check	Sum Check Bits							
Residue	Residue Bits							
FCS	FCS High Byte							
FCS	FCS Low Byte							
Closing Flag	0	1	1	1	1	1	1	0

X=Reserved

1. OPENING FLAG: As defined in the HDLC specification, used to synchronize and separate transmissions.

2. NODE ADDRESS: The address of this Node.

3. ACTIVITY SEEN: Whenever a transition on a link is detected at a port, whether enabled or not, the corresponding bit in the byte is set to a 0. These bits remain set until a clear status command is received in a valid input frame.

4. TRANSMISSION ERRORS: Whenever a Node detects a transmission error this bit is set for the corresponding port. These bits remain set until a clear status command is received in a valid input frame.

5. VALID FRAME SEEN: Whenever a frame is seen without transmission errors the corresponding port bit is set. These bits remain set until a clear status command is received in a valid input frame.

6. NODE ERRORS SEEN: Whenever a frame is received addressed to this node without transmission errors but with format errors it will not be honored by this node, the bit corresponding to the port upon which it was received will be set. These bits remain set until a clear status command is received in a valid input frame.

7. NODE VALID FRAME: Whenever a node responds to an input frame the corresponding port bit in this byte is set. This bit is set before a response transmission and cleared after the response transmission if a clear status command is received.

8. NODE PORT CONFIGURATION: This byte is normally set to the present state of the port enable register. However, if the input transmission had requested a change of port configuration for this transmission only (ENB ONCE bit set), then the byte is set to the state to which the node will revert after this transmission.

9. MESSAGE SUM CHECK: The contents of this byte is calculated such that an add, modulo 256,of the Address byte, Activity Seen byte, Transmission Errors byte,Valid Frame Seen byte, Node Errors Seen byte, Node Valid Frame byte, Node Port Configuration byte and this byte will yield a result of zero. It is computed by the Node to enable the receiving site to detect errors.

10. FCS: The FCS bytes are a cyclic redundancy calculation performed by the HDLC transmitter and appended to the end of the frame.

11. CLOSING FLAG: The closing flag is the frame terminator.

Output Frame From Message Buffer

An output frame from the message buffer is intended to be used as a test tool. The output frame information field contains the number of bytes specified in the counter starting at the address in the Address Register. The counter and Address Register must have been initialized prior to a request. The values in these registers remain unchanged until they are rewritten. A byte count of zero will result in 256 bytes being transmitted. The output frame will take the following form.

1. Opening Flag

2. Contents of Address specified by the Address Register

3. Contents of Address specified by the Address Register + 1

4. •

5. •

6. Contents of Address Specified by the Address Register + Counter

7. Residue bits

8. FCS

9. FCS

10. Closing Flag

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA CR-181678	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle

Advanced Information Processing System:
Input/Output Network Management Software

5. Report Date

May 1988

6. Performing Organization Code

7. Author(s)

Gail Nagle,
Linda Alger, and
Alexander Kemp

8. Performing Organization Report No.

CSDL-R-2039

10. Work Unit No.

506-46-21-05

9. Performing Organization Name and Address

The Charles Stark Draper Laboratory, Inc.
555 Technology Square
Cambridge, MA 02139

11. Contract or Grant No.

NAS1-17666

12. Sponsoring Agency Name and Address

National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23665-5225

13. Type of Report and Period Covered

Contractor Report

14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor: Felix L. Pitts

This report was prepared for NASA Langley Research Center under contract NAS1-17666.

16. Abstract

The Advanced Information Processing System (AIPS) uses a damage and fault tolerant network to allow communication between its General Purpose Processors (GPCs) and its I/O devices. Although the network performs exactly like a bus, it is far more reliable and damage tolerant than a linear bus. Because of the richness of interconnections between the nodes which make up the bus, a faulty component can be identified and the network reconfigured so as to isolate the failed part and restore full communication to all non-failed components.

The I/O Network Manager is the software process responsible for establishing and maintaining the communication path between processors and attached I/O devices. The methodology used in the design of the I/O Network Manager calls for a statement of the functional requirements of this software process, followed by the software specifications of the various parts of this module and their interaction with other AIPS software components. All data items are fully specified and structured flow charts are generated for all the logic required by the specifications. The final step is the implementation of the I/O Network Manager logic as an Ada language program. This report covers all phases of the design process with some additional information about the hardware used in the I/O Networks of the AIPS Proof-of-Concept System.

17. Key Words (Suggested by Author(s)) Redundancy Management, Network Manager, Fault and Damage Tolerant I/O Network, Reconfigurable Bus, Reliable Communication, Fault Isolation, Fault Analysis, Autonomous I/O	18. Distribution Statement Unclassified-Unlimited Subject Category 62

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of pages 293	22. Price A13